

HOCHSCHULE RHEIN-WAAL
&
FLUXANA GMBH & CO. KG

BACHELOR'S THESIS

**Development of an automated powder
dosing system using a 6-DOF
collaborative robotic arm (cobot)**

by investigating the influence of vibration, angle of dosing,
and rotational speed to the mass flow of the powder.

Author:

Abdelrahman MOSTAFA
Matriculation No. 29528

Supervisor:

Prof. Dr. Ronny HARTANTO
Dr. Rainer SCHRAMM

*A thesis submitted in fulfillment of the requirements
for the Bachelor degree of Science*

in the

Mechatronic Systems Engineering
Faculty of Technology & Bionics

October 27, 2023

Declaration of Authorship

I, **Abdelrahman MOSTAFA**, declare that this thesis titled, "Development of an automated powder dosing system using a 6-DOF collaborative robotic arm (cobot)" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

HOCHSCHULE RHEIN-WAAL

Abstract

Faculty of Technology & Bionics
Research & Development at Fluxana GmbH & Co. KG

Bachelor of Science

**Development of an automated powder dosing system using a 6-DOF
collaborative robotic arm (cobot)**

by **Abdelrahman MOSTAFA**

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor...

Contents

Declaration of Authorship	iii
Abstract	vii
Acknowledgements	ix
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	1
1.3 Thesis Structure	1
I Basics of Robotics, ROS, and Powder Dosage	3
2 Basics of Robotics	5
2.1 Types of Robots	6
2.2 Robot Components	7
2.2.1 Link	8
2.2.2 Joint	8
2.2.3 Manipulator	8
2.2.4 Wrist	8
2.2.5 End-Effector	8
2.2.6 Actuator	8
2.2.7 Sensors	8
2.2.8 Controller	8
2.3 Robot Kinematics	8
2.4 Robot Control (Software)	8
3 Robot Operating System ROS	9
3.1 Linux for Robotics	9
3.1.1 What Is Ubuntu? and Why for Robotics?	9
3.1.2 Ubuntu File Structure	10
3.2 Philosophy Behind ROS	11
3.2.1 Peer-to-Peer	12
3.2.2 Tool-Oriented	12
3.2.3 Multilingual	12
3.2.4 Thin	12
3.2.5 Open Source	13
3.3 Preliminaries	13
3.3.1 ROS-Graph [25]	13
3.3.2 Roscore [24]	15
3.3.3 catkin, Workspaces, and ROS Packages	16
3.4 ROS Master Communication	17

3.4.1	Publishers-Subscribers	18
3.4.2	Services	18
3.4.3	Actions	18
3.5	MoveIt! [30]	18
4	Basics of Powder Dosing	19
4.1	Affecting Parameters	19
II	Experimental Set-up, Methodology, and Results	21
5	Experimental Set-up	23
5.1	Frame	23
5.2	Crucibles	25
5.3	Balance Device	26
5.4	Ned2 Collaborative Robot	26
5.4.1	Hardware Configurations	26
Cobot's Gripper	29	
5.4.2	Software Tools	29
5.5	Precision Validation	30
5.6	Vibration Motor	30
6	Methodology	31
6.1	Sequence Logic	31
6.2	Each State of the State Diagram	31
7	Evaluation & Results	33
7.1	Evaluation	33
8	Conclusion & Future Work	35
8.1	Future Work	35
A	Frequently Asked Questions	37
B	FX_ROS.py Library in Python	39
C	Frame Technical Drawing	47
D	Balance Technical Drawing	53
E	Glass and Metal Crucibles Technical Drawing	57
	Bibliography	61

List of Figures

2.1 Examples of a humanoid and a collaborative robot.	6
3.1 Ubuntu file system structure	10
3.2 Graphical representation of a ROS system	14
3.3 Roscore Connections with the nodes	15
3.4 Master Name Service Example [23]	17
5.1 The Full Framework of System	23
5.2 Demonstration of the parts of the framework.	24
5.3 The glass crucible with its hangers.	25
5.4 The metal crucible and its hanger.	25
5.5 The balance with its metal plates and rubber buffers.	26
5.6 Ned2 cobot provided by Niryo [7].	26
5.7 The detailed workspace of Ned2 cobot.	27
5.8 The development phase of the gripper.	29
5.9 The main tasks of the gripper.	29

List of Tables

2.1	Robots Classification	7
5.1	Technical Specification of Ned2 [6]	28

Chapter 1

Introduction

1.1 Motivation

1.2 Objectives

1.3 Thesis Structure

Part I

Basics of Robotics, ROS, and Powder Dosage

Chapter 2

Basics of Robotics

As an academic field, robotics emerges as a relatively youthful discipline, characterized by profoundly ambitious objectives, the most paramount of which is the creation of machines capable of emulating human behavior and cognitive processes. This quest to engineer intelligent machines inherently compels us to embark on a journey of self-exploration. It prompts us to scrutinize the intricacies of our own design—why our bodies possess the configurations they do, how our limbs synchronize in movement, and the mechanisms behind our acquisition and execution of intricate tasks. The realization that the fundamental inquiries in robotics are intrinsically linked to inquiries about our own existence forms a captivating and immersive aspect of the robotics pursuit. [15]

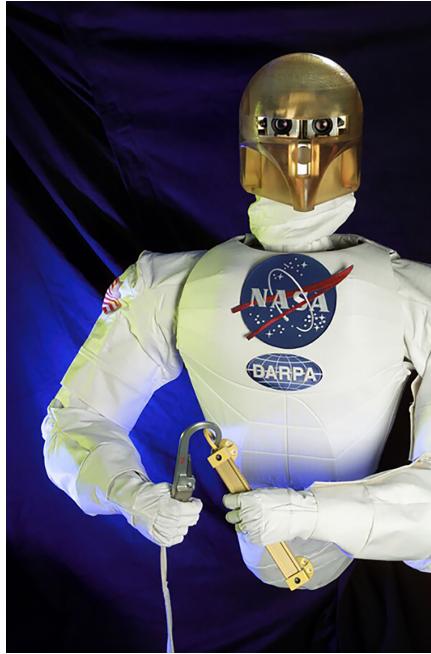
Definition ➤

Robotics is the scientific field dedicated to the study of robots—machines capable of autonomous operation, carrying out various tasks without direct human intervention. [14]

While science fiction often envisions robots in humanoid or android forms, real-world robots, especially those designed for industrial applications, typically deviate from human physical resemblance. These robots typically comprise three fundamental components: a mechanical structure, often represented by a robotic arm, enabling physical interaction with the robot's environment or itself; sensors that collect data on various physical attributes such as sound, temperature, motion, and pressure; and a processing system that interprets data from the robot's sensors, providing instructions for task execution.

It's worth noting that certain devices, like web-crawling search engine bots that systematically explore the internet to collect information on links and online content, may lack physical mechanical elements. Nonetheless, they are still classified as robots because they exhibit the ability to perform repetitive tasks autonomously.

This chapter delves into an exploration of various robot classifications, delving into the foundational principles of mechanics and kinematics. It also scrutinizes the intricacies of planning and control within the context of collaborative robots (cobots). The knowledge presented in this chapter draws significant inspiration from two primary sources, namely '**Modern Robotics**' [15] by Kevin M. Lynch and Frank C. Park, and '**Theory of Applied Robotics**' [9] authored by Professor Reza N. Jazar. For a deeper understanding of these topics, I encourage you to refer to these texts.



(a) NASA Robonaut [1] [2]



(b) Universal Robot (UR20) [31]

FIGURE 2.1: Figure (a) illustrates an instance of a humanoid robot developed by NASA, while Figure (b) exemplifies a cobot.

2.1 Types of Robots

Across diverse industries, robotics solutions have emerged as catalysts for heightened productivity, elevated safety standards, and increased operational adaptability. Organizations at the vanguard of innovation are discerning forward-looking applications of robotics that yield palpable and quantifiable outcomes. Intel collaborates closely with manufacturers, system integrators, and end-users, actively contributing to the realization of robots that deliver impactful, human-centered results.

According to an article from Intel regarding the classification of robots [8], the current generation of robots has been categorized into six distinct groups.

Autonomous Mobile Robots (AMRs) [5] AMRs navigate their environments and make rapid decisions on the fly. These robots employ advanced technologies like sensors and cameras to gather data from their surroundings. Equipped with onboard processing capabilities, they analyze this data and make well-informed decisions—whether it involves avoiding an approaching human worker, selecting the exact parcel to pick, or determining the suitable surface for disinfection. These robots are self-sufficient mobile solutions that operate with minimal human intervention. [26]

Automated Guided Vehicles (AGVs) [33] While AMRs navigate their surroundings autonomously, AGVs typically operate along fixed tracks or predetermined paths and frequently necessitate human supervision. AGVs find extensive application in scenarios involving the transportation of materials and goods within controlled settings like warehouses and manufacturing facilities.

Humanoids [12] While numerous mobile humanoid robots could, in a technical sense, be classified as Autonomous Mobile Robots (AMRs), this categorization

primarily applies to robots fulfilling human-centric roles, frequently adopting human-like appearances. These robots leverage a similar array of technological components as AMRs to perceive, strategize, and execute tasks, encompassing activities such as offering navigational assistance or providing concierge services.

Hybrids [29] Diverse categories of robots are frequently integrated to engineer hybrid solutions that possess the capacity to execute intricate operations. For instance, the fusion of an AMR with a robotic arm can yield a versatile system tailored for the handling of packages within a warehouse environment. As functionalities are amalgamated within single solutions, there is a concurrent consolidation of computational capabilities.

Articulated Robots [28] Commonly referred to as robotic arms, are designed to replicate the versatile functions of the human arm. These systems typically incorporate a range of rotary joints, varying from two to as many as ten. The inclusion of additional joints or axes equips these robotic arms with a wider range of motion capabilities, rendering them particularly well-suited for tasks such as arc welding, material manipulation, machine operation, and packaging.

Cobots [18] Collaborative Robots, commonly referred to as cobots, are engineered with the specific purpose of working in tandem with, or directly alongside, human operators. Unlike many other categories of robots that function autonomously or within strictly segregated workspaces, cobots share work environments with human personnel to enhance their collective productivity. Their primary role often involves the removal of manual, hazardous, or physically demanding tasks from daily operations. In certain scenarios, cobots are capable of responding to and learning from human movements, further enhancing their adaptability.

The initial four robots fall under the category of mobile robots, possessing the capability to navigate within their surroundings, while the latter two are categorized as stationary robots, as detailed in table 2.1 below.

TABLE 2.1: Robots Classification.

Mobile	Stationary
AMRs	
AGVs	Articulated robots
Humanoids	Cobots
Hybrids	

Within the scope of this paper, our exclusive focus will be on **cobots** [18]. Across all the experiments conducted in this study, a cobot (Ned2, detailed and described in chapter 5) has been consistently utilized.

2.2 Robot Components

In our study, we establish a kinematic model for a robotic manipulator, which is essentially a multi-body system comprising interconnected rigid bodies. These bodies

are connected through revolute or prismatic joints, enabling relative movement. We employ principles of rigid body kinematics to elucidate the relative motions between these interconnected bodies.

It's imperative to note that a comprehensive robotic system encompasses not only the manipulator or rover but also components such as the wrist, end-effector, actuators, sensors, controllers, processors, and software. [9]

2.2.1 Link

In the realm of robotics, each individual rigid component within a robot that possesses the capacity to move concerning all other components is formally known as a 'link.' This terminology accommodates various descriptions, including 'bar,' 'arm,' or any object deemed equivalent to a link in the context of robot mechanics. A robot arm or link, in essence, represents a solid, rigid element capable of relative motion when compared to the other links within the robotic structure.

Moreover, when we encounter two or more linked components that are entirely constrained in terms of relative movement, they are collectively regarded as a 'compound link,' forming a unified and motionally inseparable entity within the robot's framework.

2.2.2 Joint

2.2.3 Manipulator

2.2.4 Wrist

2.2.5 End-Effector

2.2.6 Actuator

2.2.7 Sensors

2.2.8 Controller

2.3 Robot Kinematics

2.4 Robot Control (Software)

Chapter 3

Robot Operating System | ROS

Definition

ROS serves as an open-source, meta-operating system designed to cater to the requirements of your robot. Much like traditional operating systems, it offers essential functionalities such as hardware abstraction, precise control over low-level devices, integration of commonly-used features, seamless message exchange between processes, and streamlined package management. Furthermore, ROS equips you with a comprehensive set of tools and libraries to facilitate tasks like code acquisition, compilation, development, and execution across diverse computing environments. [22]

In this chapter, we will delve into the foundational elements of the ROS framework. We will begin by exploring the relevance and basics of Linux and Ubuntu in the context of ROS. Following that, we will delve into the philosophical underpinnings of ROS, its master communication structure, and how these elements relate to topics, services, and actions. Finally, we will provide an explanation of the motion planning framework known as MoveIt.

3.1 Linux for Robotics

Linux is a free, open-source operating system that includes several utilities that will significantly simplify your life as a robot programmer. As will be shown in the upcoming sections, ROS (Robot Operating System) is based on a Linux system. All commands and concepts explained here are taken from the Linux tutorial made by the University of Surrey. [10]

3.1.1 What Is Ubuntu? and Why for Robotics?

Ubuntu, accessible at www.ubuntu.com, stands as a widely acclaimed Linux distribution rooted in the Debian architecture (source: <https://en.wikipedia.org/wiki/Debian>). Notably, it's freely available and open source, permitting extensive customization for specific applications. Ubuntu boasts an extensive software repository, comprising over 1,000 software components, encompassing essentials such as the Linux kernel, GNOME/KDE desktop environments, and a suite of standard desktop applications, including word processing tools, web browsers, spreadsheets, web servers, programming languages, integrated development environments (IDEs), and even PC games. Versatile in its deployment, Ubuntu can operate on both desktop and server platforms, accommodating architectures like Intel x86, AMD-64, ARMv7,

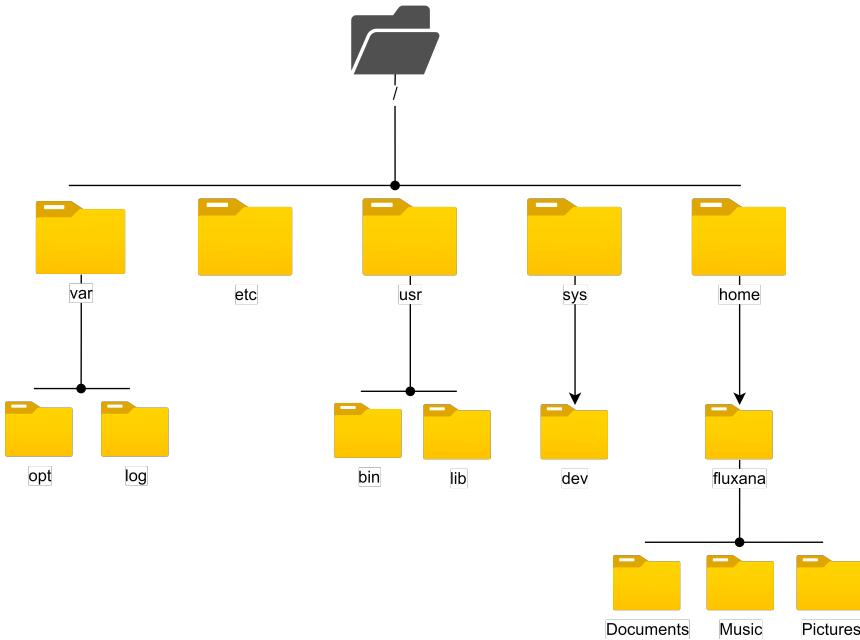


FIGURE 3.1: Ubuntu file system structure

and ARMv8 (ARM64). Canonical Ltd., headquartered in the UK (www.canonical.com), provides substantial backing to Ubuntu. [13]

In the realm of robotics, software stands as the nucleus of any robotic system. An operating system serves as the foundation, facilitating seamless interaction with robot actuators and sensors. A Linux-based operating system, such as Ubuntu, offers unparalleled flexibility in interfacing with low-level hardware while affording provisions for tailored OS configurations tailored to specific robot applications. Ubuntu's merits in this context are manifold: it exhibits responsiveness, maintains a lightweight profile, and upholds stringent security measures. Additionally, Ubuntu boasts a robust community support ecosystem and a cadence of frequent releases, ensuring its perpetual relevance. It also offers long-term support (LTS) releases, guaranteeing user assistance for up to five years. These compelling attributes have cemented Ubuntu as the preferred choice among developers in the Robot Operating System (ROS) community. Indeed, Ubuntu stands as the sole operating system that enjoys comprehensive support from ROS developers. The Ubuntu-ROS synergy emerges as the quintessential choice for programming robots. [10]

3.1.2 Ubuntu File Structure

Similar to the 'C drive' in a Windows operating system, Linux incorporates a dedicated storage area for its system files, known as the root file system. This root file system is established during the Ubuntu installation process, with the assignment of '/' as its designated mount point. For a visual representation of the Ubuntu file system architecture, refer to Figure 3.1.

The following describes the uses of each folder in the file system:

- */bin and /sbin*: These directories house essential system applications, akin to the 'C:/Windows' folder in Windows.
- */etc*: Within this directory, system configuration files are stored.

- */home/yourusername*: Equivalent to the 'C:/Users' directory in Windows, this directory serves as the user's home.
- */lib*: Similar to '.dll' files in Windows, the '/lib' directory contains library files.
- */media*: This directory serves as the mount point for removable media.
- */root*: The '/root' directory contains files associated with the root user, who holds administrative privileges in the Linux system.
- */usr*: Pronounced 'user,' the '/usr' directory hosts a majority of program files, akin to 'C:/Program Files' in Microsoft Windows.
- */var/log*: Within this directory, you'll find log files generated by various applications.
- */home/yourusername/Desktop*: The location for Ubuntu desktop files.
- */mnt*: Mounted partitions are accessible in this directory.
- */boot*: This directory stores essential files required for the boot process.
- */dev*: Linux device files are located here.
- */opt*: The '/opt' directory serves as the designated location for optionally installed programs. (For instance, ROS is installed in '/opt').
- */sys*: This directory houses files containing critical information about the system.

3.2 Philosophy Behind ROS

The philosophical objectives of ROS can be succinctly described as follows [20]:

- Decentralized collaboration: Emphasizing **peer-to-peer** interactions.
- **Tool-oriented** approach: Focusing on the development of a robust set of tools.
- **Multilingual support**: Enabling compatibility with multiple programming languages.
- **Thin** design: Prioritizing a streamlined framework.
- Openness and freedom: Being freely available and based on **open-source** principles.

To the best of our knowledge, no existing framework encompasses this specific set of design principles. This section aims to delve into these philosophies, elucidating how they have profoundly influenced the design and implementation of ROS [20].

3.2.1 Peer-to-Peer

ROS comprises a multitude of compact software components that establish connections among themselves, facilitating a perpetual exchange of messages. These messages are transmitted directly from one software component to another without the need for a centralized routing service. While this architecture may introduce added complexity to the underlying system infrastructure, it yields a critical advantage: scalability. As data volume increases, the ROS system can efficiently accommodate the rising demands without compromising its performance.

3.2.2 Tool-Oriented

Drawing inspiration from the enduring architectural principles of Unix, ROS exemplifies how intricate software systems can be constructed from an assembly of numerous small, versatile programs. Distinguishing itself from many other robotics software frameworks, ROS does not adopt a singular, integrated development and run-time environment. Instead, it delegates various tasks—such as source code tree navigation, system visualization (refer to section 3.3.1), graphical data plotting, documentation generation, data logging, and more—to discrete software programs. This decentralized approach encourages the continuous refinement and evolution of these tools. Ideally, users have the flexibility to substitute existing tools with improved implementations tailored to specific task domains.

In recent ROS iterations, there has been an advancement where multiple tools can be seamlessly integrated into single processes to enhance operational efficiency or facilitate user-friendly interfaces for operators and debugging. Nevertheless, the foundational principle persists: individual tools remain compact and versatile in nature.

3.2.3 Multilingual

Software tasks vary in their complexity and requirements, sometimes favoring ‘high-productivity’ scripting languages like Python or Ruby, while other scenarios demand the efficiency of faster languages such as C++. Preferences for languages like Lisp or MATLAB [4] also come into play, sparking debates on the best-suited language for specific tasks. Recognizing the value of these diverse perspectives and the contextual utility of languages, ROS adopts a multilingual approach. ROS permits software modules to be written in a wide array of languages, provided that a compatible client library exists. As of the current writing, ROS supports client libraries for C++, Python, LISP, Java, JavaScript, MATLAB, Ruby, Haskell, R, Julia, and others. This book predominantly utilizes the Python client library for code examples, balancing space considerations and user-friendliness. However, it’s important to note that the tasks discussed here can be achieved using any of the available client libraries.

3.2.4 Thin

The conventions within ROS promote a development approach where contributors create independent libraries and subsequently integrate them with ROS modules to facilitate message exchange. This intermediary layer serves a dual purpose: it allows for the versatile reuse of software in contexts beyond ROS, and it streamlines the process of conducting automated tests through established continuous integration tools.

3.2.5 Open Source

The core of ROS operates under the permissive BSD license [21], allowing both commercial and noncommercial utilization. ROS employs interprocess communication (IPC) to facilitate data exchange between modules. This approach enables systems built with ROS to have flexible licensing arrangements for their components. Commercial systems, for instance, may incorporate a combination of closed-source and open-source modules, while academic and hobby projects often adhere to open-source principles. Additionally, commercial product development frequently takes place within closed environments. The ROS license accommodates these diverse use cases and remains fully compliant with each scenario. [20]

3.3 Preliminaries

Before delving into ROS, it's essential to introduce the fundamental concepts that underpin this framework. ROS systems are constituted by a multitude of autonomous programs that maintain continuous communication with one another. This section provides an in-depth exploration of this architectural setup and the associated command-line tools. It further delves into the intricate aspects of ROS naming conventions and namespaces, demonstrating their role in facilitating code reusability. [19]

3.3.1 ROS-Graph [25]

One of the original challenges inspiring the creation of ROS was commonly known as the 'fetch an item' problem. This scenario involved a relatively large and complex robot equipped with various sensors, a manipulator arm, and a mobile base. In the 'fetch an item' problem, the robot's objective is to navigate a typical home or office environment, locate a specified item, and transport it to the designated location. This task led to several key observations, which subsequently became foundational design goals for ROS:

- The application task can be broken down into numerous autonomous subsystems, encompassing areas like navigation, computer vision, and grasping.
- These subsystems are adaptable for various tasks, such as security patrols, cleaning, and mail delivery, among others.
- By implementing appropriate hardware and geometry abstraction layers, the majority of application software can be made compatible with different robotic platforms.

These principles are exemplified through the core structure of a ROS system: its graphical representation. In ROS, multiple programs operate concurrently and communicate by exchanging messages. This system structure is conveniently portrayed as a mathematical graph, with nodes representing individual programs and edges indicating their communication. While Figure 3.2 illustrates a sample ROS graph from one of the early 'fetch an item' implementations, the specific details are less significant compared to the overarching concept of a ROS system as an assembly of nodes engaged in message-based communication. This representation serves as a practical framework for software development, emphasizing the modular nature of ROS programs, or 'nodes,' as integral components within a larger system.

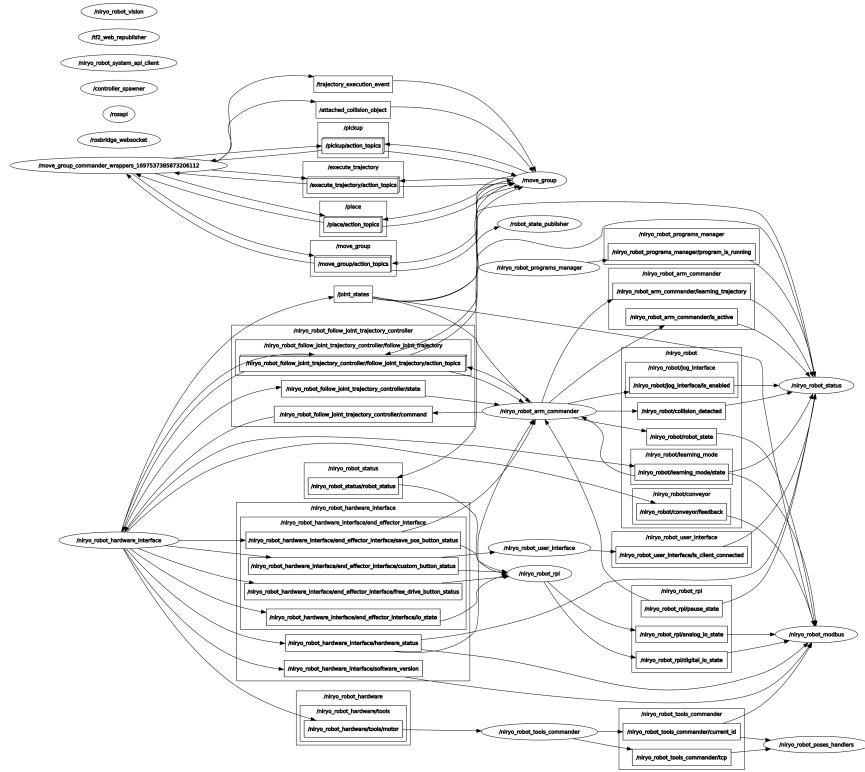


FIGURE 3.2: Graphical representation of a ROS system for ‘Niryo-Ned2 [6]’ robot—nodes/topics within the graph symbolize individual software modules, while edges denote message streams facilitating the exchange of sensor data, actuator commands, planner states, intermediate representations, and other relevant information.

In summary, within a ROS graph, a node signifies a software module engaged in message transmission, and an edge denotes the flow of messages between two nodes. While complexity can increase, nodes are typically POSIX processes, and edges are akin to TCP connections, enhancing fault tolerance as a software crash typically affects only the crashing process, leaving the rest of the graph operational. The events leading to the crash can often be reconstructed by logging messages entering a node and replaying them within a debugger at a later time.

One of the most significant advantages of a loosely coupled, graph-based architecture is the capacity to rapidly prototype complex systems with minimal or no need for additional ‘glue’ software during experimentation. Individual nodes, such as the object recognition node in a ‘fetch an item’ system, can be effortlessly replaced by launching an entirely different process that handles images and generates labeled objects. Beyond node replacement, entire segments of the graph (subgraphs) can be dynamically dismantled and substituted with other subgraphs in real time. This flexibility extends to replacing real-robot hardware drivers with simulators, swapping navigation subsystems, fine-tuning algorithms, and more. Since ROS dynamically generates the necessary network backends, the entire system fosters an interactive environment that encourages experimentation.

To this point, we have assumed that nodes discover each other, but we have not elaborated on the process. Amidst the extensive network traffic, how do nodes locate and initiate message exchange? The solution lies in a program known as ‘roscore’.

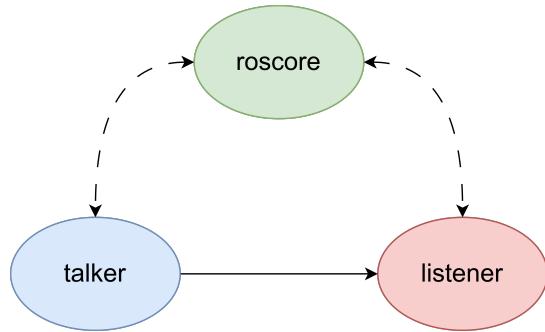


FIGURE 3.3: `roscore` establishes ephemeral connections with the other nodes in the system.

3.3.2 Roscore [24]

`roscore` serves as a vital component within the ROS ecosystem by facilitating connections between nodes to enable message transmission. During initialization, each node registers its published message streams and desired subscriptions with `roscore`, allowing it to establish direct peer-to-peer connections with other nodes participating in the same message topics. A functioning `roscore` is imperative for any ROS system since it serves as a vital reference point for nodes to discover one another.

It's important to note that while `roscore` plays a crucial role in aiding nodes in locating their peers, the actual message transmission between nodes occurs in a peer-to-peer manner. This setup can sometimes be misconstrued, especially for individuals accustomed to client/server systems from web-based backgrounds, wherein the roles of clients and servers are more distinct. The ROS architecture, however, functions as a hybrid system, integrating aspects of both client/server and fully distributed models, thanks to the central role of `roscore`, which acts as a naming service for peer-to-peer message streams.

When a ROS node initiates, it relies on the presence of an environment variable, `ROS_MASTER_URI`, which should contain a URL of the form `http://hostname:11311/`. This URL signifies the existence of a functioning `roscore` accessible on port 11311, hosted on a machine named `hostname`, which can be reached over the network.

With this information, nodes communicate with `roscore` at startup to register themselves and query for other nodes and message streams by name. Each node informs `roscore` about the messages it can provide and those it wishes to subscribe to. `roscore`, in return, supplies the necessary details about the message producers and consumers. In graphical terms, every node within the system can periodically utilize `roscore` services to identify and connect with its peers. This is illustrated by the dashed lines in Figure 3.3, signifying that in a basic two-node setup, the `talker` and `listener` nodes intermittently make service calls to `roscore` while directly engaging in peer-to-peer message exchange.

`Roscore` also serves as a parameter server, extensively utilized by ROS nodes for configuration purposes. It enables nodes to store and retrieve various data structures, including robot descriptions and algorithm parameters. To interact with the parameter server, ROS provides a command-line tool called '`rosparam`', which we will use throughout this book.

We will delve into examples of using roscore shortly. For now, it's essential to remember that roscore facilitates nodes in discovering other nodes. Before we proceed to run some nodes, it's worth understanding how ROS organizes packages and gaining some insight into the ROS build system, known as 'catkin'. [19]

3.3.3 catkin, Workspaces, and ROS Packages

Catkin serves as the ROS build system, comprising a set of tools utilized by ROS for generating executable programs, libraries, scripts, and interfaces that can be employed by other code. If you are developing your ROS code in C++, a good understanding of catkin is essential. However, since this book employs Python for its examples, we won't delve deeply into its intricacies. Nevertheless, we will explore its basic functionalities to some extent. For those interested in a more comprehensive understanding, the [catkin wiki page](#) is an excellent resource. If you are curious about why ROS has its dedicated build system, you can refer to the [catkin conceptual overview wiki page](#). To install ros-melodic catkin workspace, you can use the following command:

Command

```
sudo apt-get install ros-melodic-catkin
```

ROS Catkin Workspace

The ros workspace has several folders. Following, we will be looking at the function of each folder. [11]

src Folder The 'src' directory within the catkin workspace serves as the designated location for creating or importing new packages from repositories. It's important to note that ROS packages are only built and turned into executables when they reside in the 'src' directory. When the 'catkin_make' command is executed from the workspace directory, it scans the 'src' folder, building each package found there.

build Folder When the 'catkin_make' command is executed within the ROS workspace, the catkin tool generates certain build files and intermediate CMake cache files within the 'build' directory. These cache files play a crucial role in preventing the need to rebuild all packages each time you run 'catkin_make.' For example, if you initially build five packages and subsequently introduce a new package to the 'src' folder, only the new package will be built during the next 'catkin_make' command. This efficiency is achieved through the utilization of cache files within the 'build' directory. It's important to note that deleting the 'build' folder will trigger a complete rebuild of all packages.

devel Folder When 'catkin_make' is executed, it triggers the build process for each package, resulting in the creation of target executables if the build is successful. These executables are saved within the 'devel' folder, which contains shell script files designed to incorporate the current workspace into the ROS workspace path. Access to the packages within the current

workspace is only enabled when this script is executed. Typically, the following command is employed for this purpose.

Command ➤➤➤

```
source ~/<workspace_name>/devel/setup.bash
```

3.4 ROS Master Communication

The **ROS Master** functions as a provider of naming and registration services for the various **nodes** within the ROS system. It manages the tracking of publishers and subscribers to **topics**, as well as **services**. Essentially, the role of the Master is to facilitate the discovery of individual ROS nodes, allowing them to establish peer-to-peer communication.

Additionally, the Master offers the Parameter Server functionality.

To initiate the Master, the 'roscore' command is commonly used, which initiates the ROS Master alongside other indispensable components.

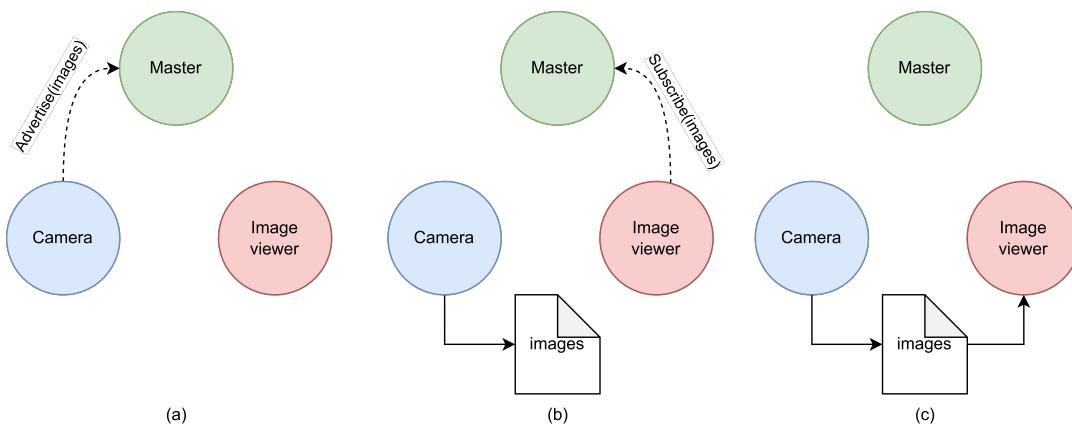


FIGURE 3.4: Master Name Service Example [23]

In the above-shown figure 3.4, let's consider a scenario involving two Nodes: a Camera node and an Image_viewer node. The typical sequence of events begins with the Camera node informing the master that it intends to publish images on the 'images' topic, as depicted in (a).

Subsequently, the Camera node commences publishing images to the 'images' topic. However, as there are no subscribers to this topic, no data is transmitted. In (b), the Image_viewer expresses its interest in subscribing to the 'images' topic to check if any images are available.

With both a publisher and a subscriber now present for the 'images' topic, the master node facilitates the awareness of Camera and Image_viewer to each other's existence, allowing them to commence the exchange of images, as illustrated in part (c) of the figure.

3.4.1 Publishers-Subscribers

The first communication

3.4.2 Services

3.4.3 Actions

3.5 MoveIt! [30]

MoveIt![3] serves as the primary software framework within the Robot Operating System (ROS) for motion planning and mobile manipulation. It has garnered acclaim for its seamless integration with various robotic platforms, including the PR2 [34], Robonaut [1], and DARPA's Atlas robot. MoveIt! is primarily coded in C++, augmented by Python bindings to facilitate higher-level scripting. Embracing the fundamental principle of software reuse, advocated for in the realm of robotics [16], MoveIt! adopts an agnostic approach towards robotic frameworks, such as ROS. This approach entails a formal separation between its core functionality and framework-specific elements, ensuring flexibility and adaptability, especially in inter-component communication.

By default, MoveIt! leverages the core ROS build and messaging systems. To facilitate effortless component swapping, MoveIt! extensively employs plugins across its functionality spectrum. This includes motion planning plugins (currently utilizing OMPL), collision detection (presently incorporating the Fast Collision Library (FCL) [17]), and kinematics plugins (employing the OROCOS Kinematics and Dynamics Library (KDL) [27] for both forward and inverse kinematics, accommodating generic arms alongside custom plugins).

MoveIt!'s principal application domain lies in manipulation, encompassing both stationary and mobile scenarios, across industrial, commercial, and research settings. For a more comprehensive exploration of MoveIt!, interested readers are encouraged to refer to [Cite here](#).

Chapter 4

Basics of Powder Dosing

4.1 Affecting Parameters

Part II

Experimental Set-up, Methodology, and Results

Chapter 5

Experimental Set-up

This chapter initiates a comprehensive exploration of the experimental setup. It commences with an overview of the workspace frame, encompassing the array of interconnected devices. These include various crucibles, the balance device, both its hardware and software components, the Ned2-cobot with its hardware configurations and software tools, precision validation procedures and concludes with an examination of the vibration motor, which is an auxiliary component integrated with the cobot.

5.1 Frame

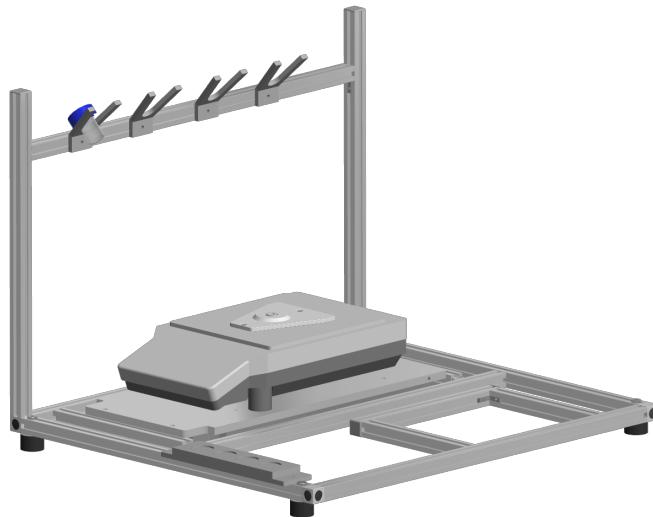


FIGURE 5.1: The Full Framework of System

In the realm of robotics, establishing a meticulously structured workspace framework is of paramount significance. This framework serves as the foundation encompassing all the requisite components with which the cobot will interact. As illustrated in Figure 5.1, this framework has been thoughtfully designed to meet these specifications.

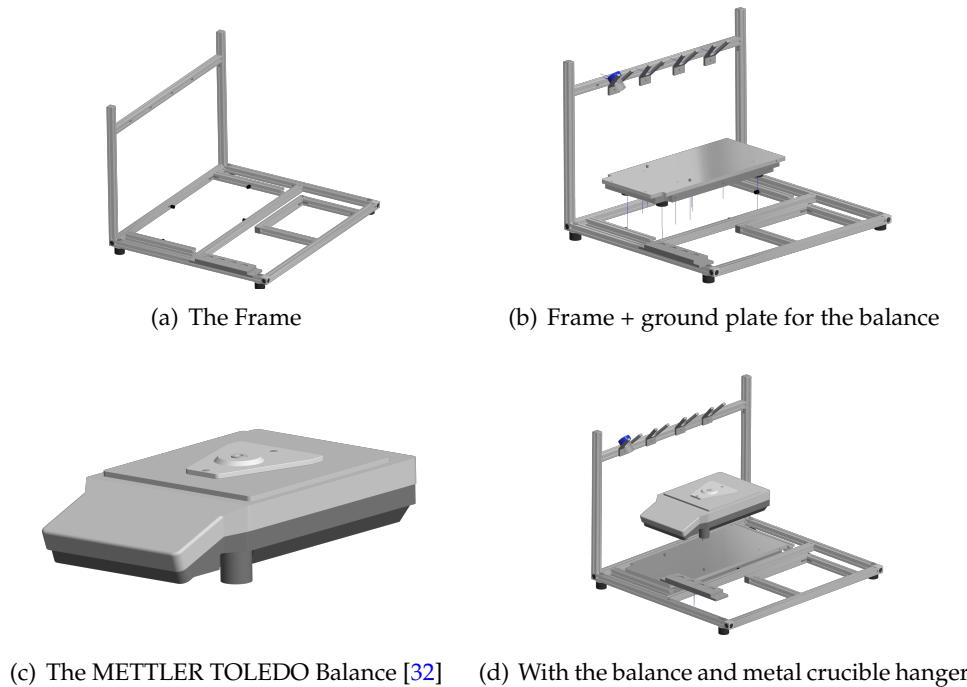


FIGURE 5.2: Demonstration of the parts of the framework.

The framework encompasses various crucial components, as detailed below:

1. **The Frame** itself, depicted in part (a) of Figure 5.2. More comprehensive technical documentation of the frame's constituent parts and dimensions can be found in Appendix C.
2. **The Ground Plate of the Balance**, featuring a rubber base, is illustrated in part (b) of Figure 5.2. This element plays a pivotal role in ensuring the stability of the balance. Given its high sensitivity and precision, it is imperative to eliminate any potential transmission of motion or vibration.
3. The **Mettler Toldedo Me Balance**, showcased in part (c) of Figure 5.2. For an in-depth understanding of the balance's specifications pertinent to our project, please refer to section 5.3.
4. **Four Glass Crucibles** and their corresponding hangers, documented comprehensively in section 5.2.
5. **Four Metal Crucibles** along with their hangers, as outlined in section 5.2.
6. The **Niryo Ned2 Cobot**, expounded upon in detail in section 5.4.
7. **Standing Rubber Buffers**, strategically incorporated to enhance the frame's stability.

Subsequent sections will delve into the intricate particulars and documentation of these essential components for our project.

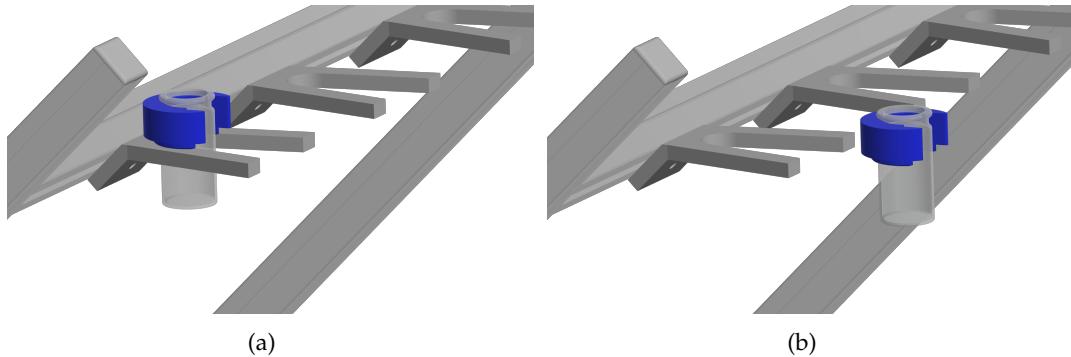


FIGURE 5.3: The glass crucible with its hangers.



(a) The technical design.

(b) An actual crucible on the hanger.

FIGURE 5.4: The metal crucible and its hanger.

5.2 Crucibles

The crucibles play a central role in our project by serving as containers for the powder to be transferred from one crucible to another.

We will utilize two distinct types of crucibles:

- **Glass Crucibles:** These crucibles will predominantly be employed for powder dosing purposes.

In the frame, we have incorporated four hangers for the glass crucibles, which can be observed in Figure 5.1. Additionally, the range of motion for the glass crucible is depicted in Figure 5.3. The crucible has an outer diameter of 22mm, and an inner diameter of about 17mm. More comprehensive information regarding the dimensions and technical specifications of the glass crucible can be found in Appendix E.

- **Metal Crucibles:** These crucibles will hold the powder after it has been dosed from the glass crucibles and will be utilized to measure the weight on the balance.

The hanger serves the purpose of accommodating four crucibles, and you can observe its design in Figure 5.4. It possesses an upper diameter of 37mm and a lower diameter of 25mm, as visualized in Figure 5.4(a). For a more in-depth exploration of the technical design and drawings of this component, please consult the details provided in Appendix E, where comprehensive information is available.

5.3 Balance Device



FIGURE 5.5: The balance with its metal plates and rubber buffers.

The precision balance employed in this study is a product of **METTLER TOLEDO**, a reputable German company. It offers an impressive readability of 0.1mg and a substantial weight capacity, approximately 220g, as referenced in [32]. For comprehensive insights into the specific components and technical specifications of this balance, please consult the information provided in Appendix D. The balance's operation, illustrated in Figure 5.5, involves its integration with two robust metal ground plates and the inclusion of four standing rubber buffers, which collectively contribute to the system's ensured stability during the weighing process.

5.4 Ned2 Collaborative Robot

Ned2 is a collaborative robot, often referred to as a cobot, developed by the French company Niryo [7]. This particular cobot has been purpose-built for educational and research applications, serving as a valuable tool for the development of proof of concepts and experimental work. In the context of this research, the Ned2 cobot played a pivotal role in conducting experiments.

The forthcoming sections delve into an in-depth examination of the hardware specifications and software options offered by the Ned2 cobot.



FIGURE 5.6: Ned2 cobot provided by Niryo [7].

5.4.1 Hardware Configurations

Ned2 is a six-axis collaborative robot, based on open-source technologies. It is intended for education, research and Industry 4.0." [6]

Incorporating the same aluminum framework as its predecessor, Ned2 maintains its commitment to meeting your exacting standards in terms of durability, precision, and repeatability (with an accuracy, and a repeatability of 0.5 mm).

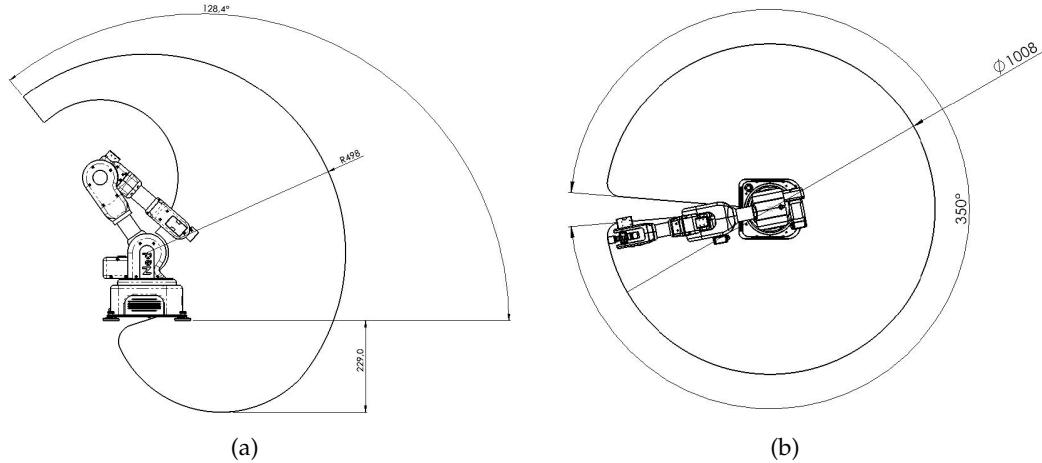


FIGURE 5.7: The detailed workspace of Ned2 cobot.

Ned2 operates on the Ubuntu 18.04 platform and utilizes the ROS Melodic framework, capitalizing on the capabilities of the **Raspberry Pi 4**. This high-performance **64-bit ARM V8 processor**, coupled with **4GB of RAM**, empowers Ned2 to deliver enhanced performance.

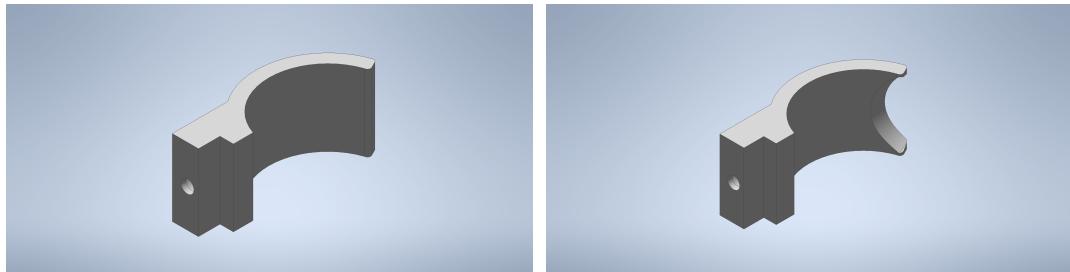
This iteration of Ned2 introduces advanced servo motors equipped with Silent Stepper Technology, significantly reducing the operational noise of the robot. The technical specifications of Ned2 are described as shown in table 5.1 below.

Furthermore, it's crucial for each robot to operate within a well-defined workspace that allows for unrestricted movement. In our case, the workspace of Ned2, as depicted in Figure 5.4 (a) and (b), has been meticulously documented in accordance with Niryo company specifications [6]. Our frame was meticulously designed and engineered to align with the defined workspace of the cobot.

Parameters	Value
Weight (kg)	7
Payload (g)	300
Reach (mm)	440
Degree of freedom	6 rotating joints
Joints range (rad)	$2,949 \leq Joint1 \leq 2,949$ $-2,09 \leq Joint2 \leq 0,61$ $-1.34 \leq Joint3 \leq 1,57$ $-2,089 \leq Joint4 \leq 2,089$ $-1,919 \leq Joint5 \leq 1.922$ $-2,53 \leq Joint6 \leq -2,53$
Joints speed limit (rad/s)	$Joint1 \leq 0.785$ $Joint2 \leq 0.5235$ $Joint3 \leq 0.785$ $Joint4 \leq 1.57$ $Joint5 \leq 1.57$ $Joint6 \leq 1.775$
TCP max speed (mm/s)	468
Repeatability (mm)	+/- 0,5
Footprint (mm)	200x200
Power supply	Input: AC100-240V / 50-60Hz, 2,5A Output: DC 12V - 7A ; 5V - 7A
I/O power supply	5V
Inputs/Outputs Control panel	Digital input x1 Digital output x1
Robot interface	USB2.0 x2 USB3.0 x2 ETHERNET GIGABIT x1
Communication	Modbus TCP (master) TCP/IP
Materials	Aluminum ABS-PC (injection moulding)
Collision detection	Accelerometer & gyroscope in the control panel
Certification	CE Conformity

TABLE 5.1: Technical Specification of Ned2 [6]

Cobot's Gripper

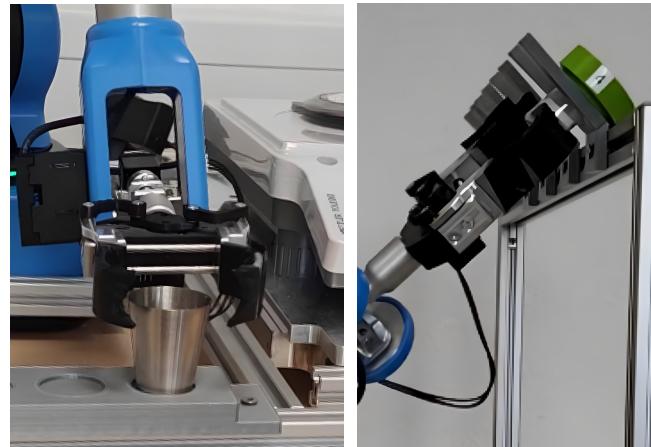


(a) The Initial phase of developing a gripper for the metal crucible only.

(b) The final phase of developing the gripper for both crucibles used in the research.

FIGURE 5.8: The development phase of the gripper.

While it's common to use general-purpose grippers such as those available from Niryo company, our research and development application necessitated a customized gripper. We have engineered a specialized gripper tailored to the unique demands of our project. In Figure 5.8, we can see the development phase of the gripper in order to fit the application for both glass and metal crucibles. The main tasks that the gripper is going to be used are holding the metal crucible from the middle, as shown in part (a) Figure 5.9, and holding the glass crucible from the bottom by the gripper's tip, as shown in part (b) Figure 5.9



(a) The initial task. (b) The second task.

FIGURE 5.9: The main tasks of the gripper.

5.4.2 Software Tools

Ned2 represents a collaborative robot, hinging on the Ubuntu 18.04 platform and **ROS Melodic**—a widely adopted open-source solution in the field of robotics. Leveraging ROS, Ned2 offers an extensive array of libraries that empower users to create a wide spectrum of programs, from the simplest to the most intricate, thus ensuring adaptability to diverse operational requirements [6].

5.5 Precision Validation**5.6 Vibration Motor**

Chapter 6

Methodology

6.1 Sequence Logic

6.2 Each State of the State Diagram

Chapter 7

Evaluation & Results

7.1 Evaluation

This is just a box!

Chapter 8

Conclusion & Future Work

small description of what I have done.. What are my final findings and thoughts...

The future work, what to come.

8.1 Future Work

```
1 #!/usr/bin/env python
2 import rospy
3 from std_msgs.msg import Int32
4 rospy.init_node('topic_publisher')
5 pub = rospy.Publisher('counter', Int32)
6 rate = rospy.Rate(2)
7 count = 0
8
9 while not rospy.is_shutdown():
10     pub.publish(count)
11     count += 1
12     rate.sleep()
```

LISTING 8.1: Some Python code

Appendix A

Frequently Asked Questions

Appendix B

FX_ROS.py Library in Python

```
1 #!/usr/bin/env python
2 import tf
3 import time
4 import sys
5
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 import rospy
10 #from ActionClient import ActionClient
11 from sensor_msgs.msg import JointState
12 from niryo_robot_arm_commander.srv import GetFK, GetFKRequest,
13     GetJointLimits, JogShift, JogShiftRequest, JogShiftResponse
14 #from niryo_robot_msgs.srv import SetBool, SetBoolRequest, SetInt,
15     SetIntRequest, Trigger
16
17 #from niryo_robot_arm_commander.msg import ArmMoveCommand,
18     RobotMoveGoal, RobotMoveAction
19
20 from moveit_msgs.srv import GetPositionFK, GetPositionIK
21
22 from std_msgs.msg import Header
23 from moveit_msgs.msg import RobotState as RobotStateMoveIt
24
25 from geometry_msgs.msg import Pose
26 import geometry_msgs
27 from niryo_robot_msgs.msg import RobotState
28 import moveit_commander
29 import moveit_msgs.msg
30 import actionlib
31
32 #robot = moveit_commander.RobotCommander()
33 #scene = moveit_commander.PlanningSceneInterface()
34 #global arm
35 def Connect_to_arm():
36     global arm
37     try:
38         arm = moveit_commander.move_group.MoveGroupCommander("arm")
39     except:
40         raise RuntimeError
41
42 def Call_Aservice(service_name, type, request_name=None, req_args=None):
43     """
44     Call a ROS service.
45
46     Parameters:
47     .....
48     service_name: str
49     type: srv
50
51     Returns:
52     .....
53     result: type
54
55     Raises:
56     ...
57     """
58
59     if request_name is None:
60         request_name = type
61
62     if req_args is None:
63         req_args = {}
64
65     if service_name is None:
66         raise ValueError("Service name must be specified")
67
68     if type is None:
69         raise ValueError("Service type must be specified")
70
71     if not isinstance(service_name, str):
72         raise TypeError("Service name must be a string")
73
74     if not isinstance(type, str):
75         raise TypeError("Service type must be a string")
76
77     if not isinstance(request_name, str):
78         raise ValueError("Request name must be a string")
79
80     if not isinstance(req_args, dict):
81         raise ValueError("Request arguments must be a dictionary")
82
83     if request_name not in dir(type):
84         raise ValueError(f"Service type '{type}' does not have a '{request_name}' method")
85
86     if not callable(getattr(type, request_name)):
87         raise ValueError(f"Service type '{type}' does not have a valid '{request_name}' method")
88
89     if req_args:
90         if not all(isinstance(k, str) and k != "service_name" for k in req_args):
91             raise ValueError("Request arguments must be strings and not contain 'service_name' key")
92
93     if req_args:
94         if not all(isinstance(v, type) for v in req_args.values()):
95             raise ValueError("Request arguments must be of type 'type'")
96
97     if req_args:
98         if not all(isinstance(v, type) for v in req_args.values()):
99             raise ValueError("Request arguments must be of type 'type'")


100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2288
2289
2290
2291
2292
2293
2294
2295
2295
2296
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
23
```

```

46     request_name: None (srv)
47     req_args: None (dictionary) ex. {'positon': 210, 'id': 11, 'value':
48         False}
49     should_return ?: None (int) >> is set to 1, if you want to return
50     the response of the service.
51
52     Returns:
53     .....
54     If should_return is set to 1, the function is going to return the
55     response of the service.
56     Otherwise, the function should only call the service to do a
57     certain action with no return.
58     """
59     try:
60         rospy.wait_for_service(service_name, 2)
61     except (rospy.ServiceException, rospy.ROSEException) as e:
62         rospy.logerr("Timeout and the Service was not available : " +
63         str(e))
64         return RobotState()
65
66     try:
67         service_call = rospy.ServiceProxy(service_name, type)
68
69         if request_name == None:
70             response = service_call()
71         else:
72             request = request_name()
73             for key, value in req_args.items():
74                 #print("f{key} = {value}")
75                 method = setattr(request, key, value)
76             response = service_call(request)
77
78     except rospy.ServiceException as e:
79         rospy.logerr("Failed to call the Service: " + str(e))
80         return 0
81
82     return response
83
84 def Subscribe(topic_name, type, msg_args):
85     """Subscribe to a certain topic.
86
87     Parameters:
88     .....
89     topic_name: str
90     type: srv
91     msg_args: list >> list of strings, which contains the arguments
92     that we need to read from the topic.
93
94     Returns:
95     .....
96     Return a list of the read values from each argument.
97     If we have only one argument, it returns the value of this argument
98     only, not a list.
99     """
100
101
102     #rospy.init_node('FX_ROS_Subscriber')
103
104     try:
105         msg = rospy.wait_for_message(topic_name, type, 2)
106     except:
107         rospy.logerr("Timeout and the Topic Did not receive any
108         messages")
109         return 0

```

```

101
102     value = []
103
104     if len(msg_args) == 1:
105         value = getattr(msg, msg_args[0])
106     else:
107         for i in msg_args:
108             value.append(getattr(msg, i))
109
110     return value
111
112 def Get_joints():
113     """return a tuple of 6 values for each joint from 1 till 6"""
114
115     joints_values = Subscribe('/joint_states', JointState, ["position"])
116
117     return joints_values
118
119 def get_pose():
120     """Gets the pose values from the robot_state topic.
121     Return:
122     .....
123     a list of two dictionaries, the first is positions (x,y,z),
124     whereas the second is the rpy (roll, pitch, yaw)
125     """
126
127
128     return Subscribe('/niryo_robot/robot_state', RobotState, ['position',
129     'rpy'])
130
131 def get_pose_list():
132     """Use get_pose() function to get the pose, and turn it into a list
133     .
134     Return:
135     .....
136     A list of floats >>> [x, y, z, roll, pitch, yaw]
137     """
138
139     pose = get_pose()
140     position = pose[0]
141     rpy = pose[1]
142
143     return [position.x, position.y, position.z, rpy.roll, rpy.pitch,
144     rpy.yaw]
145
146 def Get_FK_Niryo(joints):
147     """Give the the joints' values to the forward kinematics service
148     provided by Niryo, and get the pose coordinations.
149     """
150
151     fk_service = '/niryo_robot/kinematics/forward'
152     return Call_Aservice(fk_service, GetFK, GetFKRequest, {'joints':
153     joints}, should_return=1).pose
154
155 def FK_Moveit(joints):
156     """Get Forward Kinematics from the MoveIt service directly after
157     giving joints
158     :param joints
159     :type joints: list of joints values
160     :return: A Pose state object
161     @example of a return
162
163     position:

```

```

158     x: 0.278076372862
159     y: 0.101870353599
160     z: 0.425462888681
161 orientation:
162     x: 0.0257527874589
163     y: 0.0122083384395
164     z: 0.175399274203
165     w: 0.984084775322
166
167     """
168     rospy.wait_for_service('compute_fk', 2)
169     moveit_fk = rospy.ServiceProxy('compute_fk', GetPositionFK)
170
171     fk_link = ['base_link', 'tool_link']
172     header = Header(0, rospy.Time.now(), "world")
173     rs = RobotStateMoveIt()
174
175     rs.joint_state.name = ['joint_1', 'joint_2', 'joint_3', 'joint_4',
176     'joint_5', 'joint_6']
177     rs.joint_state.position = joints
178
179     reponse = moveit_fk(header, fk_link, rs)
180
181     return reponse.pose_stamped[1].pose
182
183 def Jog_shift(joints_or_pose, axis, value):
184     """Use the service jog_shift_commander to shift one axis.
185     Paramters:
186     .....
187     joints_or_pose: int >>> 1 for joints_shift, and 2 for pose_shift
188     axis: int >>> (1,2,3,4,5,6) = (x,y,z,roll,pitch,yaw)
189     value: float >> the value for which you want to shift the Jog axis.
190
191     Returns: None
192     .....
193     """
194
195     axis -= 1
196     name = "/niryo_robot/jog_interface/jog_shift_commander"
197     shift_values = [0, 0, 0, 0, 0, 0]
198     shift_values[axis] = value
199
200     req_arg = {'cmd': joints_or_pose, 'shift_values': shift_values}
201
202     Call_Aservice(name, JogShift, JogShiftRequest, req_arg)
203
204 def Move_pose_axis(axis, new=None, add=None, arm_speed=None):
205     """You should either put a value to add or new, not both.
206
207     Parameters:
208     .....
209     * axis: str -> (x, y, z, roll, pitch, or yaw)
210     * new: float -> The new coordination you want to give to a certain
211       axis.
212       "new" will always overwrite the value of the axis.
213     * add: float -> the value in meters or radians you want to add to a
214       certain axis.
215     * arm_speed: float (optional) -> between 0 and 1. (0,1]
216     Returns: None
217     .....
218     """
219     FK = get_pose()

```

```

218     axeses = ['x', 'y', 'z']
219
220     pose = Pose()
221     p_goal = pose.position
222     orn_goal = pose.orientation
223
224     p_current = FK[0]
225
226     rpy_current = FK[1]
227
228     if add:
229         if axis.lower() in axeses:
230             current_value = getattr(p_current, axis)
231             setattr(p_current, axis, current_value+add)
232         else:
233             current_value = getattr(rpy_current, axis)
234             setattr(rpy_current, axis, current_value+add)
235     if new:
236         if axis.lower() in axeses:
237             setattr(p_current, axis, new)
238         else:
239             setattr(rpy_current, axis, new)
240
241     p_goal.x = p_current.x
242     p_goal.y = p_current.y
243     p_goal.z = p_current.z
244
245     orn_goal.x, orn_goal.y, orn_goal.z, orn_goal.w = tf.transformations
246     .quaternion_from_euler(rpy_current.roll,rpy_current.pitch,
247     rpy_current.yaw)
248
249     arm.set_pose_target(pose)
250
251     if arm_speed:
252         set_speed(arm_speed)
253     arm.go(wait=True)
254
255     arm.stop()
256     arm.clear_pose_targets()
257
258 def Move_to_pose(pose_values, arm_speed=None):
259     """Move to a given pose values.
260     Parameters:
261     .....
262
263     pose_values: list or tuple -> [x, y, z, roll, pitch, yaw]
264     arm_speed: float (optional) -> between 0 and 1. (0,1)
265     """
266
267     pose = Pose()
268     p_goal = pose.position
269     orn_goal = pose.orientation
270
271     p_goal.x = pose_values[0]
272     p_goal.y = pose_values[1]
273     p_goal.z = pose_values[2]
274
275     roll = pose_values[3]
276     pitch = pose_values[4]
277     yaw = pose_values[5]

```

```

278     orn_goal.x, orn_goal.y, orn_goal.z, orn_goal.w = tf.transformations
279     .quaternion_from_euler(roll,pitch,yaw)
280
281     #arm.set_goal_tolerance(0.001)
282     if arm_speed:
283         set_speed(arm_speed)
284     arm.set_pose_target(pose)
285     arm.go(wait=True)
286
287     arm.stop()
288     arm.clear_pose_targets()
289
290 def move_to_joints(joints, arm_speed=None):
291     """Move to a given joint values.
292     Parameters:
293     .....
294     joints: list or tuple -> [joint1, joint2, joint3, joint4, joint5,
295     joint6]
296     arm_speed: float (optional) -> between 0 and 1. (0,1]
297     """
298     joints_limits = Get_Joints_limits()
299
300     for i in range(6):
301         if joints_limits.joint_limits[i].max < joints[i] or joints[i] <
302             joints_limits.joint_limits[i].min:
303             print("Joint{} = {}, which is out of limit!".format(i+1,
304             joints[i]))
305             print("Joint{} can not be more than {} neither less than {}"
306             ".format(i+1, joints_limits.joint_limits[i].max, joints_limits.
307             joint_limits[i].min))")
308             return
309         else:
310             pass
311
312     #arm.set_joint_value_target(joints)
313     if arm_speed:
314         set_speed(arm_speed)
315     arm.go(joints, wait=True)
316
317     arm.stop()
318
319 def Move_joint_axis(axis, new=None, add=None, arm_speed=None):
320     """You should either put a value to add or new, not both.
321
322     Parameters:
323     .....
324     * axis: int -> the number of the joint that you want to move
325
326     * new: float -> The new coordination you want to give to a joint (axis).
327         "new" will always overright the value of the axis.
328     * add: float -> the value in meters change in a certain joint (axis).
329     * arm_speed: float (optional) -> between 0 and 1. (0,1]
330
331     Returns: None
332     .....
333     """
334     moving_joints = list(Get_joints())
335
336     if new:
337         moving_joints[axis-1] = new

```

```

333     elif add:
334         moving_joints[axis-1] += add
335
336     joints_limits = Get_Joints_limits()
337
338     if joints_limits.joint_limits[axis-1].max < moving_joints[axis-1]
339     or moving_joints[axis-1] < joints_limits.joint_limits[axis-1].min:
340         print("The joint{} can not be more than {} neither less than {}"
341             .format(axis, joints_limits.joint_limits[axis-1].max,
342                     joints_limits.joint_limits[axis-1].min))
343         return 0
344     else:
345         pass
346
347     arm.set_joint_value_target(moving_joints)
348     if arm_speed:
349         set_speed(arm_speed)
350     arm.go(moving_joints, wait=True)
351
352     arm.stop()
353
354 def Get_Joints_limits():
355     """Getting the limits for each joint.
356
357     You can get any joint limits as following:
358
359     Get_Joints_limits().joint_limits[0 - 5].max (float)
360     Get_Joints_limits().joint_limits[0 - 5].min (float)
361     Get_Joints_limits().joint_limits[0 - 5].name (str)
362
363     Where 0 for (joint 1), and 5 for (joint 6)
364     max, min, or name would give the maximum, minimum, or name of the
365     indicated joint.
366     """
367
368     joints_limits = Call_Aservice('/niryo_robot_arm_commander/
369     get_joints_limit', GetJointLimits)
370     return joints_limits
371
372 def set_speed(speed):
373     """Set a scaling factor for optionally reducing the maximum joint
374     velocity. Allowed values are in (0,1]."""
375     arm.set_max_velocity_scaling_factor(speed)
376
377 def wait(duration):
378     """wait for a certain time.
379
380     :param duration: duration in seconds
381     :type duration: float
382     :rtype: None
383     """
384     time.sleep(duration)
385
386 def move_with_action(pose):
387     """Still under development"""
388
389     moveit_commander.roscpp_initialize(sys.argv)
390     rospy.init_node('simple_action', anonymous=True)
391
392     robot_arm = moveit_commander.move_group.MoveGroupCommander("arm")
393
394     robot_client = actionlib.SimpleActionClient('execute_trajectory',
395         moveit_msgs.msg.ExecuteTrajectoryAction)

```

```

389     robot_client.wait_for_server()
390     #rospy.loginfo('Execute Trajectory server is available for robot')
391
392     robot_arm.set_pose_target(pose)
393     #robot_arm.set_pose_target([0.29537095654868956, 4.675568598554573e
394     -05, 0.4286678926923855, 0.0017192879795506913,
395     0.0014037282477544944, 0.00016120358136762693])
396     robot_plan_home = robot_arm.plan()
397
398
399     robot_goal = moveit_msgs.msg.ExecuteTrajectoryGoal()
400     robot_goal.trajectory = robot_plan_home
401
402     robot_client.send_goal(robot_goal)
403     robot_client.wait_for_result()
404     robot_arm.stop()
405
406 def move_pose_orn(pose, arm_speed=None):
407     """Move to a given pose values, but with orientation not rpy.
408
409     Parameters:
410     .....
411     * pose: A Pose state object
412
413     example of the pose state object that should be given:
414     =====
415     position:
416         x: 0.278076372862
417         y: 0.101870353599
418         z: 0.425462888681
419     orientation:
420         x: 0.0257527874589
421         y: 0.0122083384395
422         z: 0.175399274203
423         w: 0.984084775322
424     =====
425     """
426
427     arm.set_pose_target(pose)
428     if arm_speed:
429         set_speed(arm_speed)
430     arm.go(wait=True)
431
432     arm.stop()
433     arm.clear_pose_targets()
434
435 def move_to_named_pos(position_name, arm_speed=None):
436     """Available names:
437     - 'resting'
438     - 'straight_forward'
439     - 'straight_up'
440     """
441
442     arm.set_named_target(position_name)
443     if arm_speed:
444         set_speed(arm_speed)
445     arm.go(wait=True)

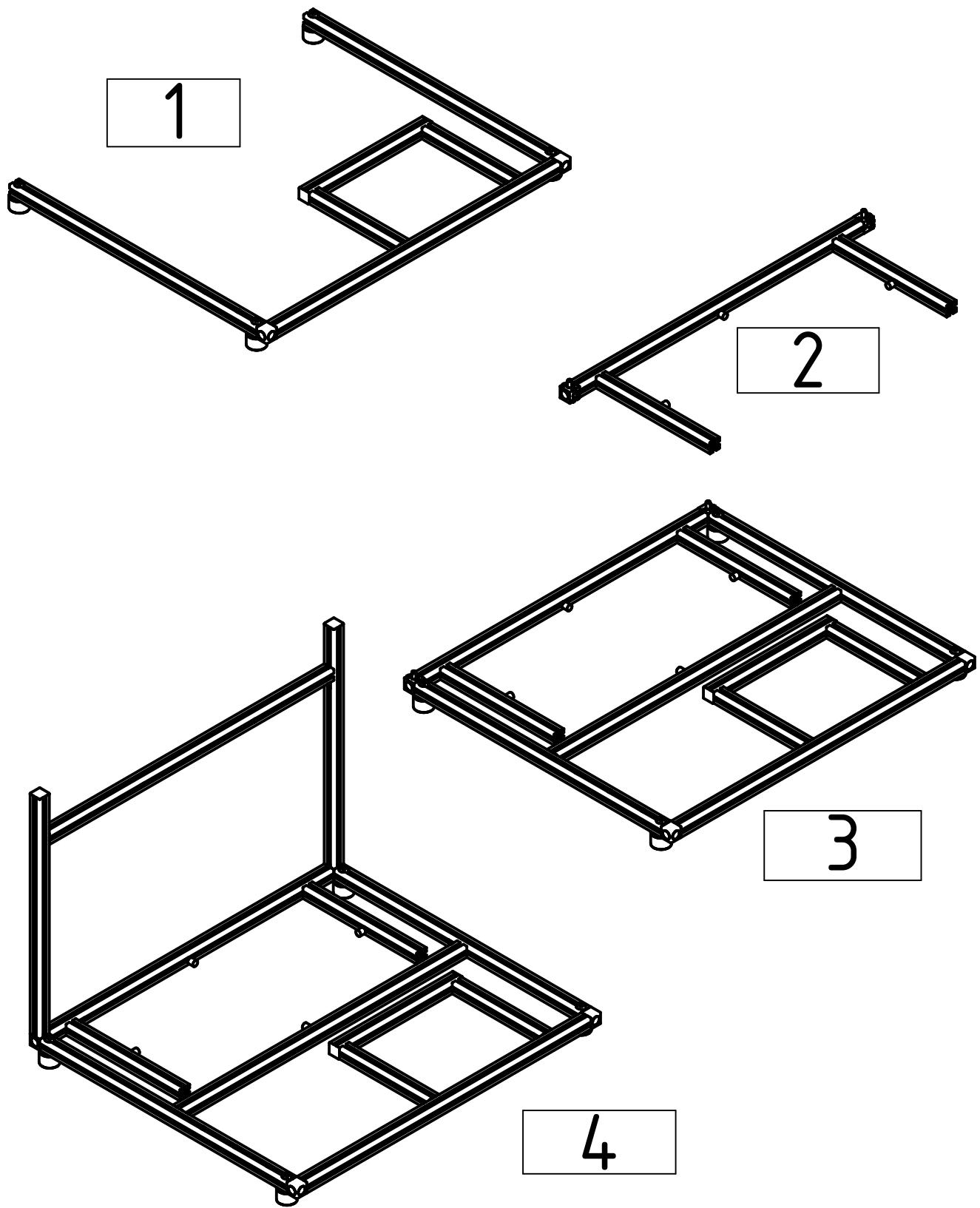
```

Appendix C

Frame Technical Drawing

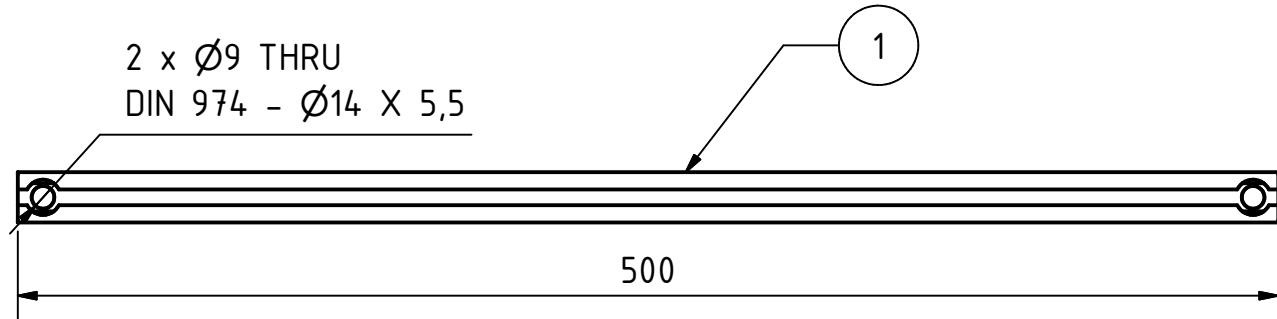
The distribution and reproduction of this drawing, the exploitation and communication of its contents are prohibited unless expressly permitted. Any infringement shall give rise to an obligation to pay damages. All rights reserved in the event of patent, utility model or design registration. Observe protection notice according to ISO 16016.

Weitergabe sowie Vervielfältigung dieser Zeichnung, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet. Zu widerhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten. Schutzvermerk nach ISO 16016 beachten.

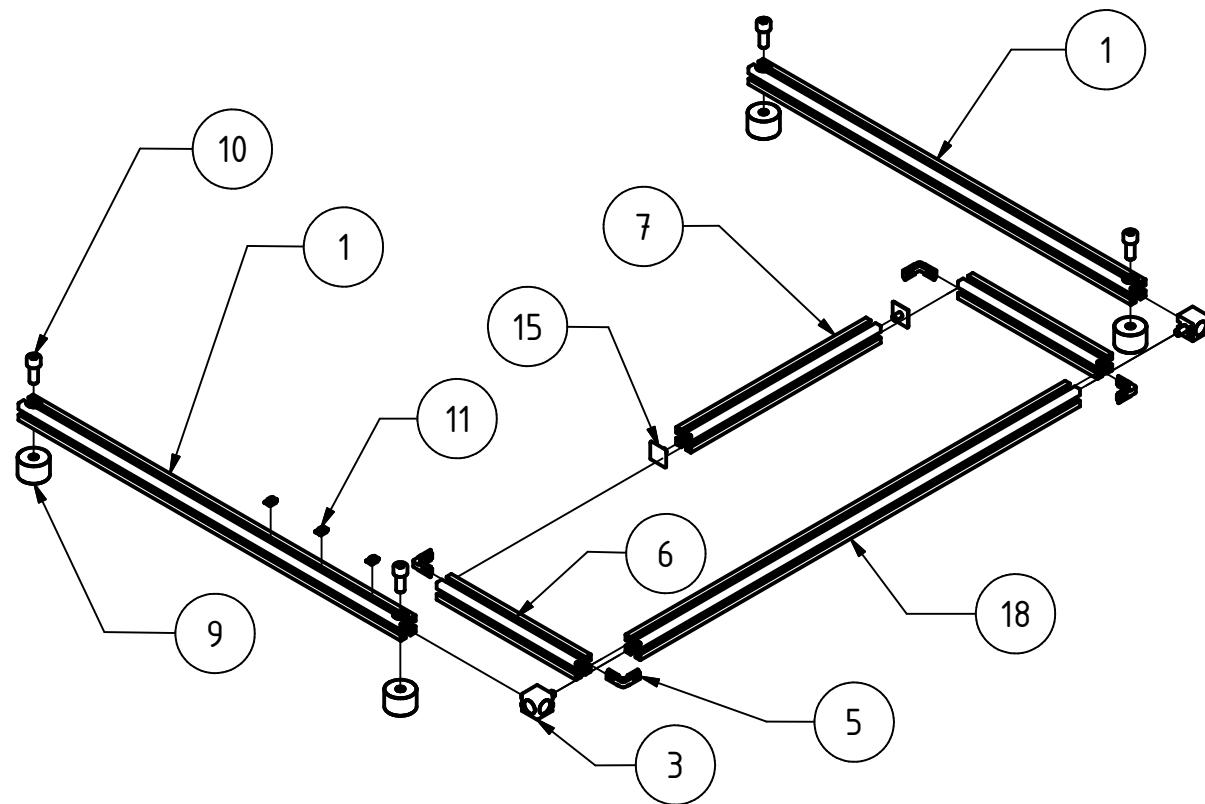


Erstellt durch: Carine Allen	Genehmigt von: 	Gewicht: N/A		
Montagebaugruppenbez.: Niryo Frame				
FLUXANA® XRF Application Solutions	Projekt:		Artikelnr.:	
Montagebaugruppenzeichnungsnr.:				
Format: A4	Maßstab: 1 : 8	Änd.:	Datum: 21/06/2023	Spr.:
		0		Blatt 1/5

Punkt 1: 2 x Bohrung im Profil 20x200x500mm



Schritt 1: Niryo holder



Erstellt durch: Genehmigt von: Gewicht:
Carine Allen N/A



Montagebaugruppenbez.: Niryo Frame

FLUXANA®
XRF Application Solutions

Projekt:

Artikelnr.:

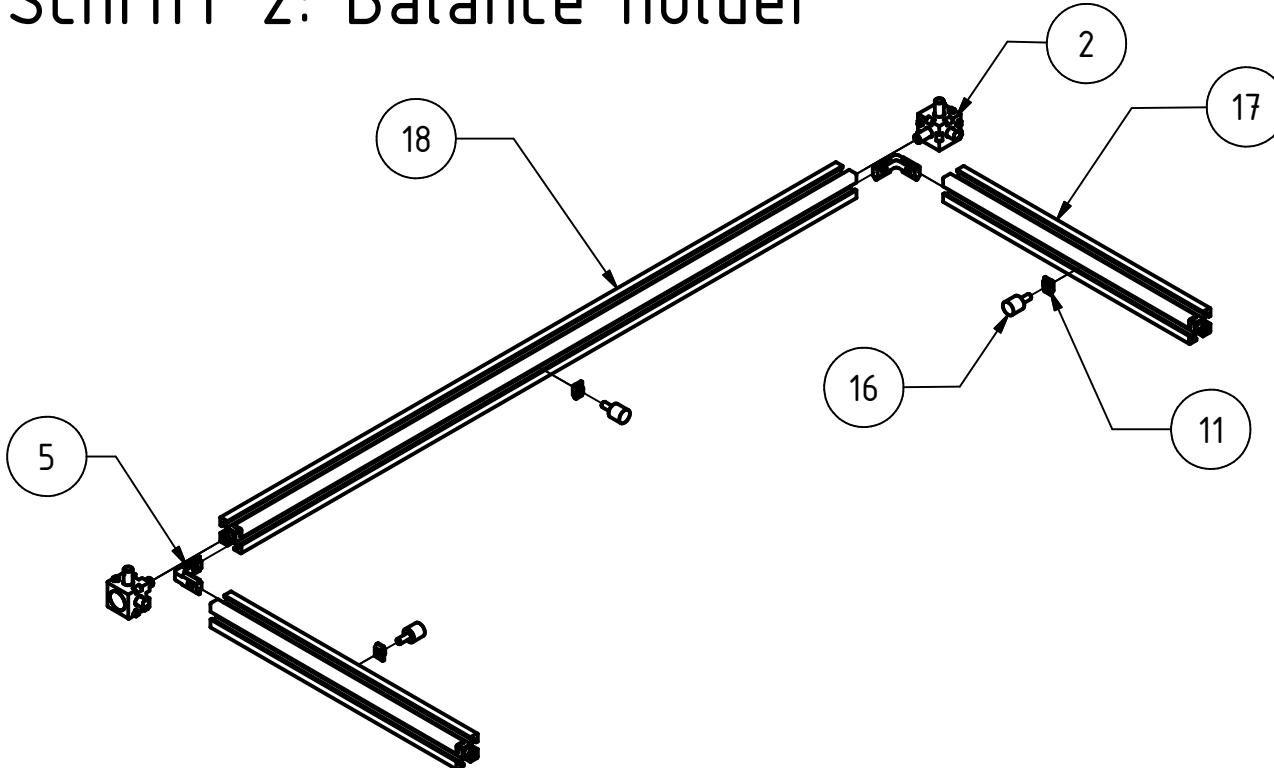
Montagebaugruppenzeichnungsnr.:

Format: A4	Maßstab: 1 : 7	Änd.: 0	Datum: 21/06/2023	Spr.: de	Blatt: 2/5
------------	----------------	---------	-------------------	----------	------------

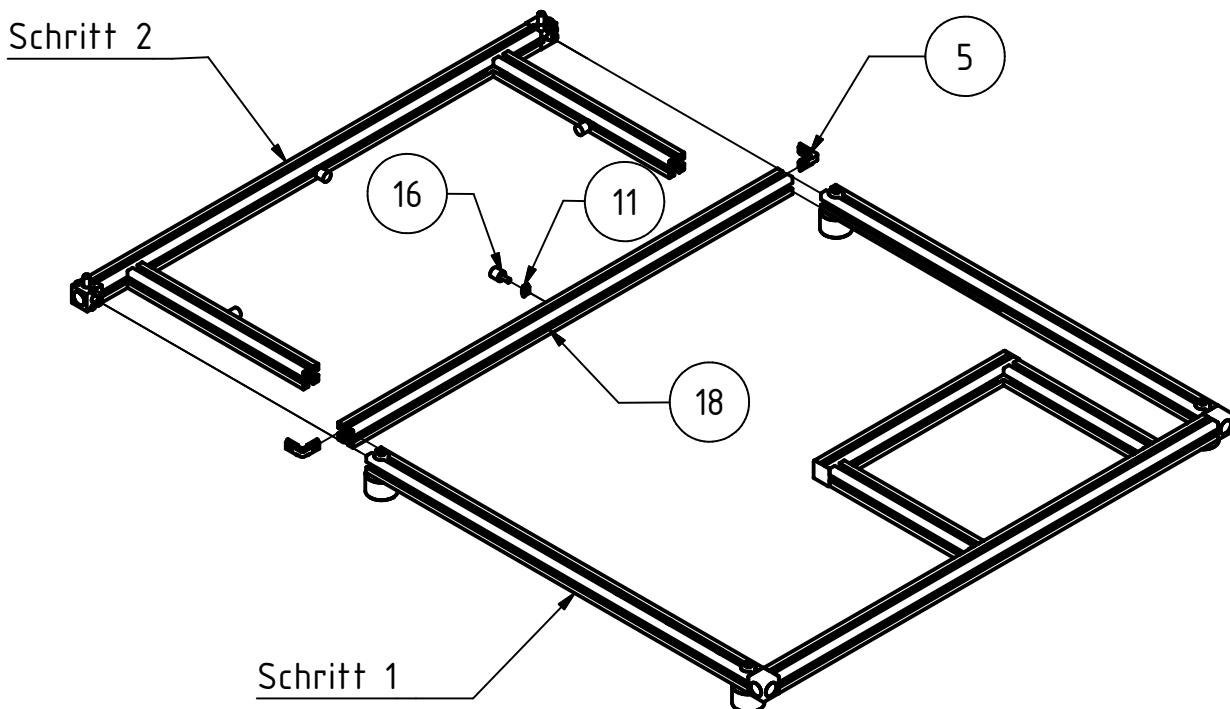
The distribution and reproduction of this drawing, the exploitation and communication of its contents are prohibited unless expressly permitted. Any infringement shall give rise to an obligation to pay damages. All rights reserved in the event of patent, utility model or design registration. Observe protection notice according to ISO 16016.

Weitergabe sowie Vervielfältigung dieser Zeichnung, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet. Zu widerhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten. Schutzvermerk nach ISO 16016 beachten.

Schritt 2: Balance holder



Schritt 3: Base frame



Erstellt durch: Carine Allen	Genehmigt von:	Gewicht: N/A		
Montagebaugruppenbez.: Niryo Frame				
FLUXANA® XRF Application Solutions	Projekt:	Artikelnr.:		
Montagebaugruppenzeichnungsnr.:				
Format: A4	Maßstab: 1 : 5	Änd.:	Datum: 21/06/2023	Spr.:
		0		Blatt 3/5

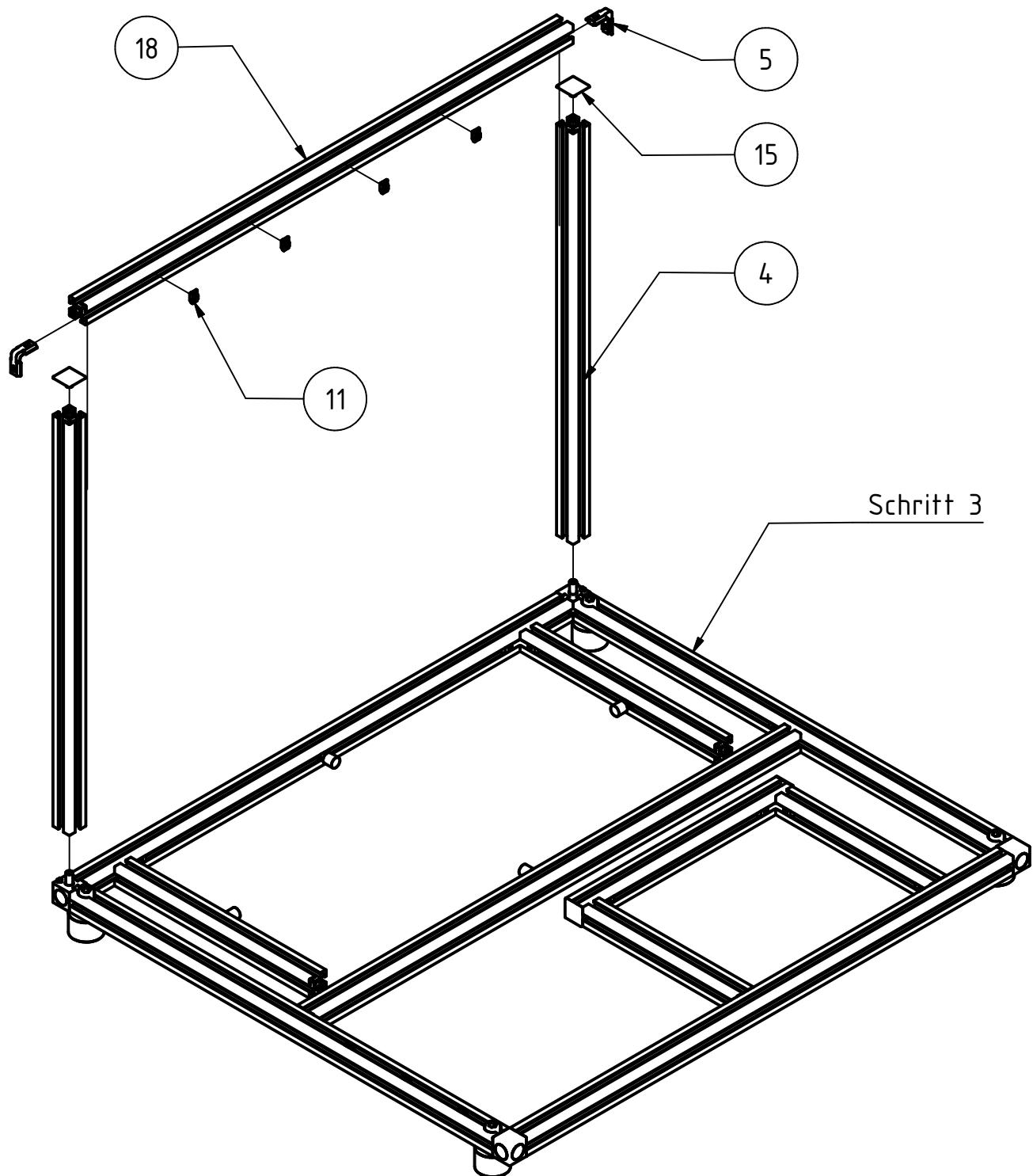
The distribution and reproduction of this drawing, the exploitation and communication of its contents are prohibited unless expressly permitted. Any infringement shall give rise to an obligation to pay damages. All rights reserved in the event of patent, utility model or design registration. Observe protection notice according to ISO 16016.

Weitergabe sowie Vervielfältigung dieser Zeichnung, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet. Zu widerhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten. Schutzvermerk nach ISO 16016 beachten.

Schritt 4: Complete frame

The distribution and reproduction of this drawing, the exploitation and communication of its contents are prohibited unless expressly permitted. Any infringement shall give rise to an obligation to pay damages. All rights reserved in the event of patent, utility model or design registration. Observe protection notice according to ISO 16016.

Weitergabe sowie Vervielfältigung dieser Zeichnung, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet. Zu widerhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten. Schutzvermerk nach ISO 16016 beachten.



Erstellt durch: Carine Allen	Genehmigt von:	Gewicht: N/A		
Montagebaugruppenbez.: Niryo Frame				
FLUXANA® XRF Application Solutions	Projekt:	Artikelnr.:		
Montagebaugruppenzeichnungsnr.:				
Format: A4	Maßstab: 1 : 5	Änd.:	Datum: 21/06/2023	Spr.:
		0		Blatt 4/5

Montagebaugruppenstückliste

Pos.	Anz.	Artikelnr.	Norm-/ BT-nr	Beschreibung
1	2	VI-0342		Strebenprofil 20x20x500mm
2	2	VI-0348		Würfelverbinder 20/3
3	2	VI-0347		Würfelverbinder 20/2
4	2	VI-0342		Strebenprofil 20x20x420mm
5	10	VI-0344		Innenwinkel 6R
6	2	VI-0342		Strebenprofil 20x20x182mm
7	1	VI-0342		Strebenprofil 20x20x248mm
8	1		BG0106	Schublade mit Waage
9	4	BO-0222		Gummipuffer 30x20mm M8x8
10	4		ISO 4762 - M8 x 20	Hexagon Socket Head Cap Screw
11	11	KT-06-0012		Hammermutter 6 M4
12	4		Becherglass_han ger	
13	1		Schnappdeckelgl as	
14	1	BO-0289	BT0953	Adapterring Schnappdeckelglas Boramat 30
15	4	KT-07-0001		Abdeckkappe grau 20x20
16	4	GR-0186		Gummipuffer D10 x H10 M4x10AES
17	2	VI-0342		Strebenprofil 20x20x230mm
18	4	VI-0342		Strebenprofil 20x20x575mm
19	3		Schnappdeckelgl as_hanger	

The distribution and reproduction of this drawing, the exploitation and communication of its contents are prohibited unless expressly permitted. Any infringement shall give rise to an obligation to pay damages. All rights reserved in the event of patent, utility model or design registration. Observe protection notice according to ISO 16016.

Weitergabe sowie Vervielfältigung dieser Zeichnung, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet. Zu widerhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten. Schutzvermerk nach ISO 16016 beachten.

Erstellt durch:	Genehmigt von:	Gewicht:	
-----------------	----------------	----------	---

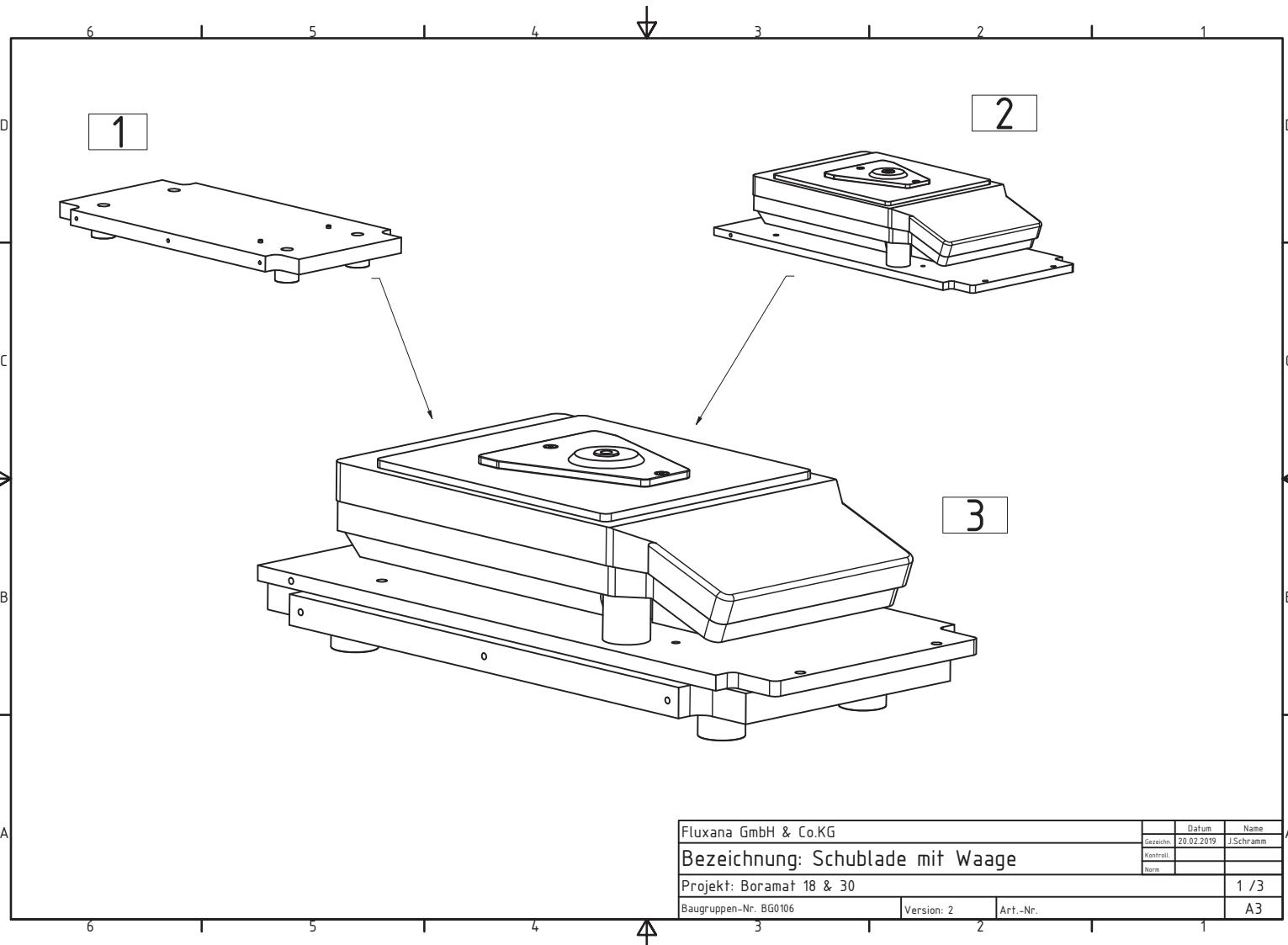
Montagebaugruppenbez.: Niryo Frame

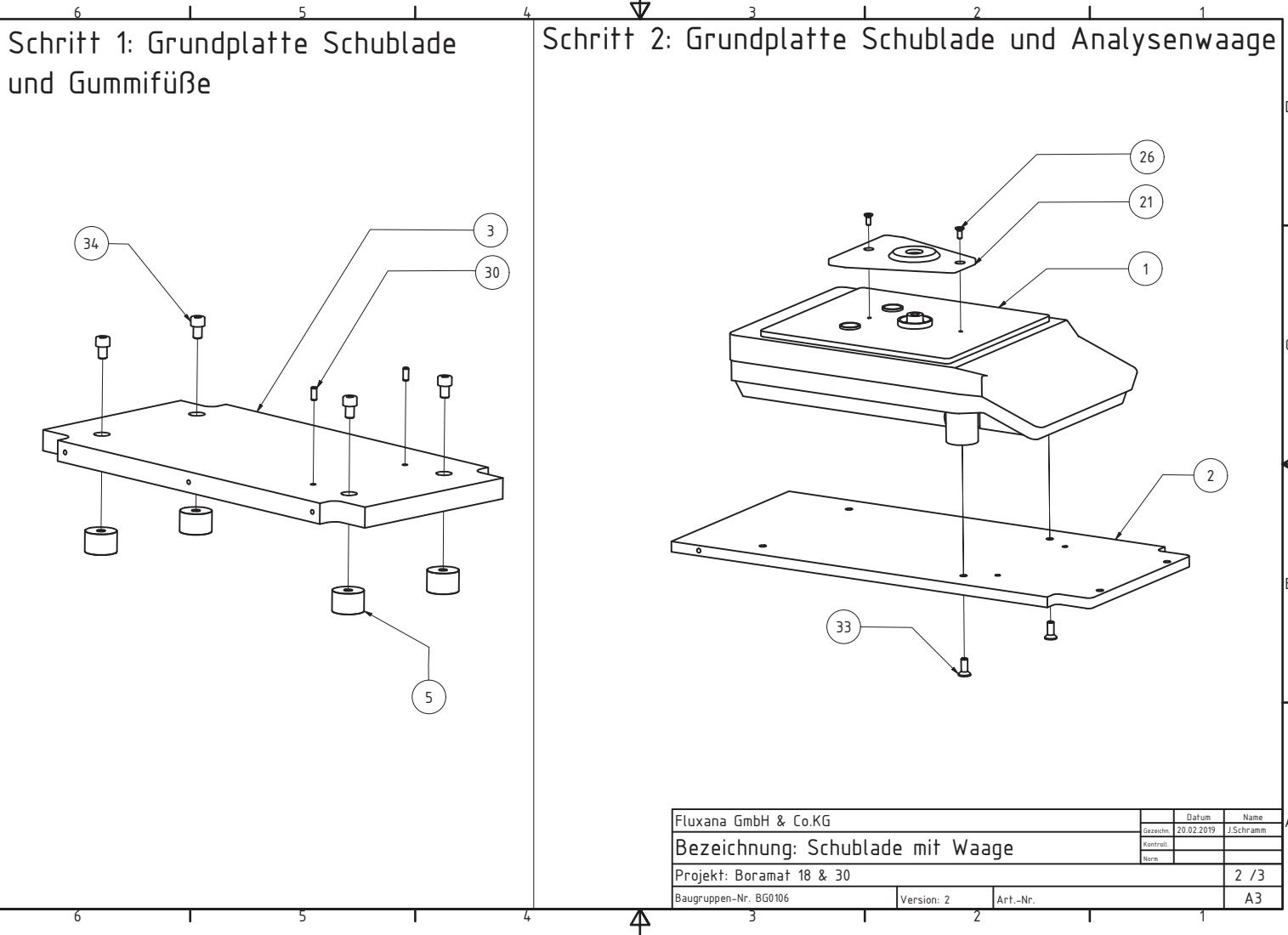
FLUXANA®
XRF Application Solutions

Projekt:	Artikelnr.:				
	Montagebaugruppenzeichnungsnr.:				
	Format:	Maßstab:	Änd.:	Datum:	Spr.:
	A4	0	21/06/2023	de	Blatt 5/5

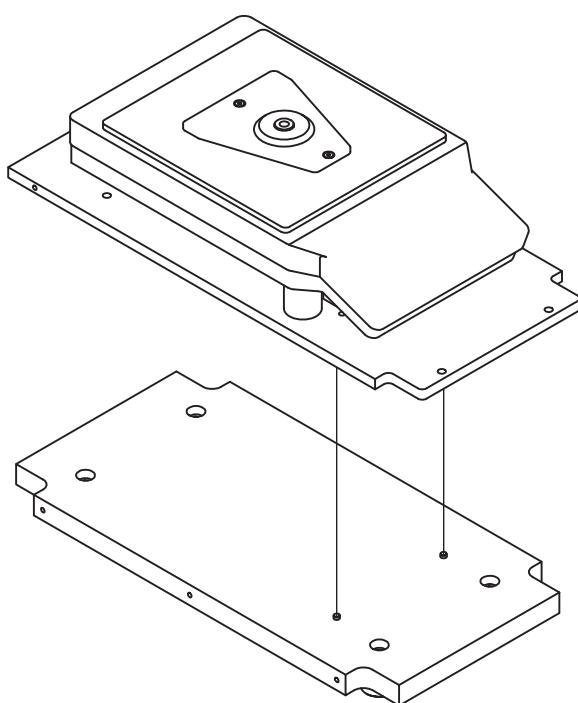
Appendix D

Balance Technical Drawing





Schritt 3: Montage der Führungsschienen



PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1		Mettler Toledo Waage
2	1	BT0413	Grundplatte Waage
3	1	BT0414	Grundplatte Schublade
5	4		Gummipuffer 30x20mm M8x8
21	1	BT0419	Schutz Waage
26	2	DIN EN ISO 10642	Senkschraube mit Inbus M4 x 10
30	2	ISO 2338	Zylinderstifte 5 m6 x 12
33	2	DIN EN ISO 10642	Senkschraube mit Inbus M6 x 16
34	4	DIN EN ISO 4762	Zylinderschraube mit Inbus M8 x 12

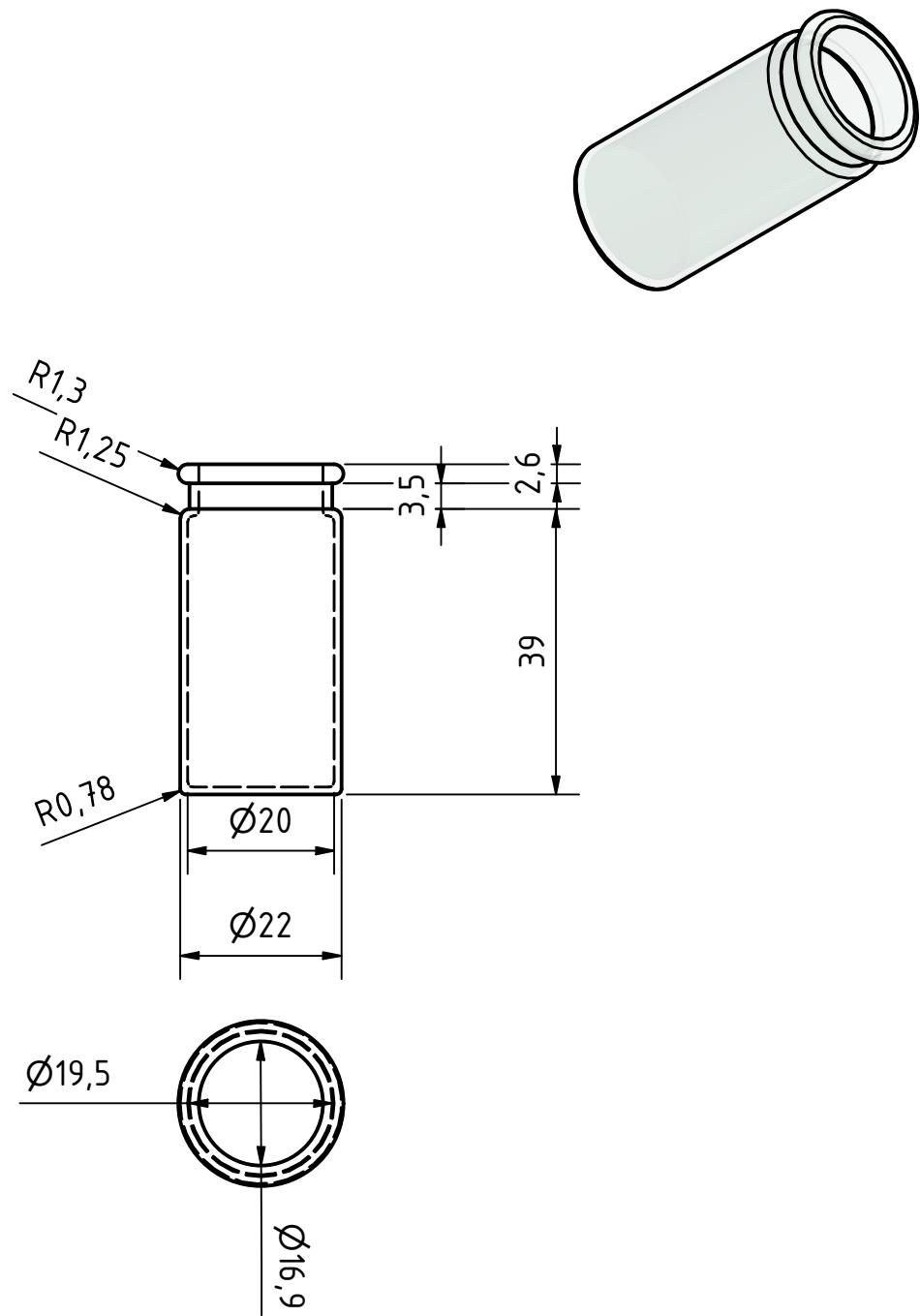
Fluxana GmbH & Co.KG	Datum	Name
Gezeichnet: 20.02.2019	J.Schramm	
Kontrolliert:		
Norm:		
Projekt: Boramat 18 & 30	3 /3	
Baugruppen-Nr. BG0106	Version: 2	Art.-Nr. A3

Appendix E

Glass and Metal Crucibles Technical Drawing

The distribution and reproduction of this drawing, the exploitation and communication of its contents are prohibited unless expressly permitted. Any infringement shall give rise to an obligation to pay damages. All rights reserved in the event of patent, utility model or design registration. Observe protection notice according to ISO 16016.

Weitergabe sowie Vervielfältigung dieser Zeichnung, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet. Zu widerhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksmustereintragung vorbehalten. Schutzvermerk nach ISO 16016 beachten.



Allgemeintoleranz: DIN ISO 2768-mK	Alle Kanten R0,1 - 0,3 entgratet		
------------------------------------	----------------------------------	--	--

Erstellt durch: Abdelrahman Mostafa	Genehmigt von: N/A	Gewicht: -	Werkstoff: Glas
--	-----------------------	---------------	--------------------

Bauteilbezeichnung:

FLUXANA®
XRF Application Solutions

Projekt:

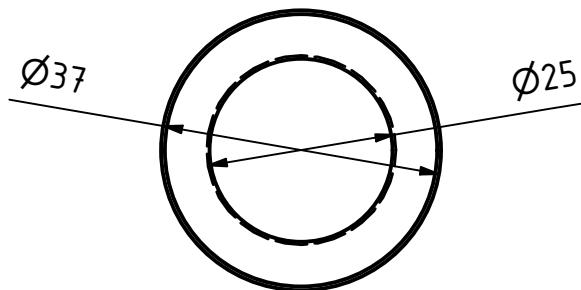
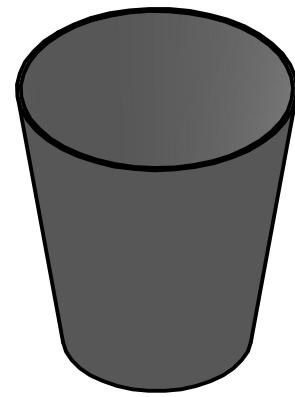
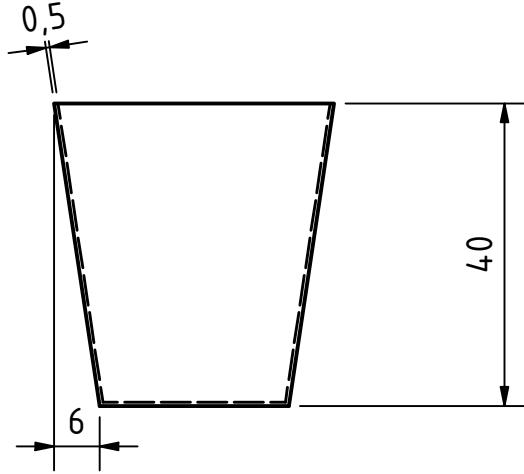
Artikelnr.:

Bauteilzeichnungsnr.: Schnappdeckelglas-1

Format: A4	Maßstab: 1 : 1	Änd.: 0	Datum: 24/10/2023	Spr.: de	Blatt: 1/1
------------	----------------	---------	-------------------	----------	------------

The distribution and reproduction of this drawing, the exploitation and communication of its contents are prohibited unless expressly permitted. Any infringement shall give rise to an obligation to pay damages. All rights reserved in the event of patent, utility model or design registration. Observe protection notice according to ISO 16016.

Weitergabe sowie Vervielfältigung dieser Zeichnung, Verwertung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich gestattet. Zu widerhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent-, Gebrauchsmuster- oder Geschmacksurteileintragung vorbehalten. Schutzzertifikat nach ISO 16016 beachten.



Allgemeintoleranz: DIN ISO 2768-mK	Alle Kanten R0,1 - 0,3 entgratet					
Erstellt durch: Abdelrahman Mostafa	Genehmigt von: N/A	Gewicht: Werkstoff: Generisch				
Bauteilbezeichnung:						
FLUXANA® XRF Application Solutions	Projekt:	Artikelnr.:				
		Bauteilzeichnungsnr.: Boromat_tiegel-1				
	Format: A4	Maßstab: 1 : 1	Änd.: 0	Datum: 24/10/2023	Spr.: de	Blatt: 1/1

Bibliography

- [1] Robert O Ambrose et al. "Robonaut: NASA's space humanoid". In: *IEEE Intelligent Systems and Their Applications* 15.4 (2000), pp. 57–63.
- [2] Tessa Brazda. *NASA Robonaut first generation*. <https://www.nasa.gov/technology/r1-the-first-generation/>. found in. 2023.
- [3] David Coleman et al. "Reducing the barrier to entry of complex robotic software: a moveit! case study". In: *arXiv preprint arXiv:1404.3785* (2014).
- [4] Peter Corke. "Integrating ros and matlab [ros topics]". In: *IEEE Robotics & Automation Magazine* 22.2 (2015), pp. 18–20.
- [5] Farbod Fahimi. *Autonomous robots*. Springer, 2009.
- [6] Marc-Henri Frouin. *Ned2 Documentation*. <https://docs.niryo.com/product/ned2/v1.0.0/en/source/introduction.html>. found in. 2022.
- [7] Marc-Henri Frouin. *Niryo Company*. <https://niryo.com>. found in. 2017.
- [8] Intel. *Types of Robots: How Robotics Technologies Are Shaping Today's World*. <https://www.intel.com/content/www/us/en/robotics/types-and-applications.html>. found in. 2023.
- [9] Reza N. Jazar. *Theory of Applied Robotics: Kinematics, Dynamics, and Control*. 3rd edition. Cham: Springer International Publishing, Imprint: Springer, 2022. DOI: [10.1007/978-3-030-93220-6](https://doi.org/10.1007/978-3-030-93220-6).
- [10] Lentin Joseph and Aleena Johny. "Getting Started with Ubuntu Linux for Robotics". In: *Robot Operating System (ROS) for Absolute Beginners: Robotics Programming Made Easy*. Springer, 2022, pp. 1–52.
- [11] Lentin Joseph and Aleena Johny. "Programming with ROS". In: *Robot Operating System (ROS) for Absolute Beginners: Robotics Programming Made Easy*. Springer, 2022, pp. 173–240.
- [12] Charles C Kemp et al. "Humanoids". In: *experimental psychology* 56 (2009), pp. 1–3.
- [13] Jay LaCroix. *Mastering Ubuntu Server: Master the art of deploying, configuring, managing, and troubleshooting Ubuntu Server 18.04*. Packt Publishing Ltd, 2018.
- [14] B.A. Lee M. "Robotics Background Information and Facts". In: *Salem Press Encyclopedia of Science* (2022). URL: <https://bvyg.short.gy/robotics-background>.
- [15] Kevin M Lynch and Frank C Park. *Modern robotics*. Cambridge University Press, 2017. URL: www.cambridge.org/9781107156302.
- [16] Alexei Makarenko, Alex Brooks, and Tobias Kaupp. "On the benefits of making robotic software frameworks thin". In: *International Conference on Intelligent Robots and Systems-Workshop for Measures and Procedures for the Evaluation of Robot Architectures and Middleware at IROS'07*. Vol. 2. 2007.

- [17] Jia Pan, Sachin Chitta, and Dinesh Manocha. "FCL: A general purpose library for collision and proximity queries". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 3859–3866.
- [18] Michael Peshkin and J Edward Colgate. "Cobots". In: *Industrial Robot: An International Journal* 26.5 (1999), pp. 335–341.
- [19] Morgan Quigley, Brian Gerkey, and William D Smart. *Programming Robots with ROS: a practical introduction to the Robot Operating System*. " O'Reilly Media, Inc.", 2015.
- [20] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. Kobe, Japan. 2009, p. 5.
- [21] Open Robotics. ROS. <https://www.ros.org/>. found in. 2015.
- [22] Open Robotics. ROS-documentation. <https://docs.ros.org/>. found in. 2015.
- [23] Open Robotics. ROS-Master. <https://wiki.ros.org/Master>. found in. 2015.
- [24] Open Robotics. roscore. <https://wiki.ros.org/roscore>. found in. 2015.
- [25] Open Robotics. rqt_graph. https://wiki.ros.org/rqt_graph. found in. 2015.
- [26] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.
- [27] Ruben Smits, H Bruyninckx, and E Aertbeliën. *KDL: Kinematics and dynamics library*. <https://www.orocos.org/kdl>. found in. 2011.
- [28] Andrew Spielberg et al. "Functional co-optimization of articulated robots". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 5035–5042.
- [29] Adam A. Stokes et al. "A Hybrid Combining Hard and Soft Robots". In: *Soft Robotics* 1.1 (2014), pp. 70–74. DOI: [10.1089/soro.2013.0002](https://doi.org/10.1089/soro.2013.0002).
- [30] Ioan A. Sucan and Sachin Chitta. MoveIt. <https://moveit.ros.org/>. found in. 2013.
- [31] Inc. Teradyne. Universal Robots. <https://www.universal-robots.com/>. found in. 2005.
- [32] METTLER TOLEDO. METTLER TOLEDO ME BALANCE. https://bvyg.short.gy/me_balance. 2020.
- [33] Günter Ullrich et al. "Automated guided vehicle systems". In: *Springer-Verlag Berlin Heidelberg. doi 10* (2015), pp. 978–3.
- [34] Keenan A Wyrobek et al. "Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot". In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 2165–2170.